REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
14/09/2011	Final Technical Report	15/05/2010-15-06/2011

4. TITLE AND SUBTITLE
(DARPA) STOCHASTIC ANALYSIS AND DESIGN OF SYSTEMS

5a. CONTRACT NUMBER
FA9550-10-C-0116

5b. GRANT NUMBER
NA

5c. PROGRAM ELEMENT NUMBER
NA

6. AUTHOR(S)
Mathew, George, A
Pinto, Alessandro

5d. PROJECT NUMBER
NA

5e. TASK NUMBER
NA

5f. WORK UNIT NUMBER
NA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
United Technologies Corporation
United Technologies Research Center
411 Silver Lane
East Hartford, CT 06118-1127

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
USAF, AFRL DUNS 143574726
AF OFFICE OF SCIENTIFIC RESEARCH
875 N. RANDOLPH ST. ROOM 3112
ARLINGTON VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
A - Approved For Public Release, Distribution is Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Analysis and design methods for stochastic hybrid dynamical systems are presented. Analysis methods include reachability analysis as well as statistical approaches. System refinement and decomposition are explored as possible approaches to deal with complexity. Findings show that analysis of this class of systems is not scalable. A promising approach is demonstrated which relies on the automatic construction of decentralized control systems using desirable properties as constraints. The synthesized system does not require to be analyzed thereby cutting the analysis effort. This approach is promising but for now limited in its use.

15. SUBJECT TERMS
Stochastic verification, hybrid systems, distributed

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Alessandro Pinto
U	U	U	UU		19b. TELEPHONE NUMBER *(Include area code)* 860 906 4690

Stochastic Analysis and Design of Systems

George A. Mathew and Alessandro Pinto
United Technologies Research Center (UTRC), Inc., Berkeley, California.
mathewga,pintoa@utrc.utc.com

Abstract

Analysis and design methods for stochastic hybrid dynamical systems are presented. Analysis methods include reachability analysis as well as statistical approaches. System refinement and decomposition are explored as possible approaches to deal with complexity. Findings show that analysis of this class of systems is not scalable. A promising approach is demonstrated which relies on the automatic construction of decentralized control systems using desirable properties as constraints. The synthesized system does not require to be analyzed thereby cutting the analysis effort. This approach is promising but for now limited in its use.

Contents

List of Figures

List of Tables

Executive Summary

Complex systems span several domains including mechanical and electrical parts as well as software controlling them. These types of systems exhibit a wide variety of time-scales: software reaction time can be as low as micro seconds while some mechanical components have reaction times spanning minutes. The external environment changes according to its own time scales. All these sub-systems (including the environment) are seldom known perfectly. Even the software components may implement randomized algorithms thereby behaving probabilistically.

These systems are known as Stochastic Hybrid Systems (SHS). They are a mix of discrete state changes occurring at some instants in time, such as state changes in a state machine, and dynamics which is governed by differential equations that model the behavior of the environment and of the electro-mechanical parts. Noise, failures, and uncertainty in the behavior of the environment make the system stochastic. As representative example, this study provides a prototypical thermal management system for aerospace applications.

Assuming that the model accurately represent the system, a useful verification tool takes the system description as input and provides answer to probabilistic queries such as the probability of a certain event to occur. The event can be defined in terms of a software state to be reached, or the value of a certain continuous variable to exceed a critical threshold. The assumption we stated at the beginning of the paragraph is strong since probability distributions of system parameters and statistics of random processes affecting the dynamic evolution of the system are typically difficult to obtain. Thus, one has to avoid undertaking the complexity of an accurate computation of such probabilities if the models are not accurate to begin with. When such probability distributions are not known, non-determinism is perhaps a better way of modeling the possible outcomes of an action in the system.

Stochastic analysis is a useful tool for designers to check properties of these types of systems. The designer is in charge of assembling the system and defining the control algorithms to achieve a certain level of performance. Another useful tool takes a partial description of the system (perhaps just the mechanical and electrical components) and a desired level of performance, and computes a control architecture that satisfies the requirements. This is referred to as the *synthesis* problem. If such tool existed, then it would be unnecessary to verify that the SHS model satisfies the requirements, simply because they were taken into account as constraints (and therefore automatically satisfied).

From the theoretical standpoint, the analysis of SHS models in the general case is an undecidable problem. This study is concerned with the practical complexity of the analysis and synthesis problems. Several techniques are reviewed and algorithms implemented to assess how memory and computational limitations constraint the complexity of the systems that can be analyzed. In the sequel, we will use the term analysis to refer to computation methods given that we do not assume any particular structure of the SHS. Analysis can be done using two classes of methods: based on reachability computation and on simulation. Methods based on reachability analysis scale poorly with the dimension of the state space, meaning the number of continuous variables that describe the state of the dynamical system. We have observed a practical limitation (on a single processor machine) to four continuous variables. Of course, this limit depends on dynamics of the system (i.e. the form of the differential equations), and on the statistics of the uncertainty. Simulation based

methods scale better but with some limitations and drawbacks. These methods cannot deal with system that exhibit non-determinism which is considered an important modeling feature for the reasons mentioned above. Moreover, the answers provided by these methods have a certain probability of being wrong.

To overcome the complexity barrier, the design flow can be structured in such a way that the system is first analyzed using abstract models, and then refined into more detailed models. The analysis conducted at the abstract level can be used to derived partitioned requirements for the sub-systems. Each sub-system can then be analyzed against the derived requirements. This method, however, requires the ability to derive probabilistic requirements for the sub-systems which come in the form of a probabilistic input-state-output relation. The second challenge is to check that the sub-systems "probabilistically" refines those requirements. Both problems are hard. This study provides some approximate methods to solve this problem. Conducting the analysis at multiple levels could in principle allow to verify system with many components as long as group of components can be conservatively modeled by one abstract component.

Given the complexity of the analysis problem, a synthesis approach has been investigated. Designing systems that are correct-by-construction can potentially cut the analysis effort altogether. The problem of deriving a decentralized control algorithm for a given mechanical system that can satisfy some probabilistic safety properties is addressed in this study. The performance of the control system and its optimal architecture are affected by the underlying computation and communication platforms. Their join optimization seems to be a hard problem. However, it is possible to divide the synthesis problem into two steps by building a suitable abstraction of the hardware platform. The control synthesis problem can therefore be addressed first. The result is an optimal decentralized control architecture and a set of requirements for the underlying computation and communication platforms. The design of the hardware platform can then be performed as a second step. The control optimization problem can be formulated as a stochastic optimization problem. Given the sparsity of the structure of the problem, it is possible to use efficient solvers even for large systems. However, some restrictions have to be placed on the type of SHS to be optimized. The more restricting ones is that transitions can only be triggered by time. Extensions are possible but need to be investigated further.

Summary of findings and recommendations. The Stochastic Hybrid System model of computation is very expressive and capable of capturing dynamical systems with discrete modes of operation evolving under uncertain conditions. Analysis of these models is hard and does not seem to be a viable method for systems of large size except when their structure leads to simplifications. The way in which a system is modeled can also render the analysis task complex. When possible, modelers should limit the number of variables used and should simplify the dynamics of the system by using discrete modes instead. Simulation methods scale better but they do not provide the same guarantees of other analysis tools, and do no allow non-determinism in the model. Correct-by-construction design could potentially eliminate the need for complex analysis, but a general approach for system synthesis as yet to appear.

Further investment is recommended in the following two areas:

- Modeling. The same system can have several representations. Moreover, domain experts are typically able to define views of the system tailored to the verification of particular properties with the same accuracy of the full model. There is a quest for verification engineers with strong domain knowledge for the development of analyzable models. Conservative abstraction methods for SHS models is also suggested as area of investment.

- Synthesis. Correct-by-construction design is a promising approach. The generality and scalability of this approach depends on the availability of a library of "control templates", namely parametrized control blocks that can be composed into a decentralized control algorithm and that lead to tractable stochastic optimization problems. A library of such templates which is expressive enough to address realistic control problem is an interesting area of investment.

Chapter 1

Abstract model of a thermal management system

We model a thermal management system at a high level of abstraction. The purpose of the model is to enable verification of stochastic properties related to fuel temperature. We model the thermal management system of a prototypical aircraft and we also build context models such as the class of possible missions to be flown, the weather conditions, and the heat loads from other sub-systems. We include several sources of uncertainty such as the mission profile and the inaccuracy of the models due to abstraction. We use a mix of dynamical and steady state models which lead to a system of differential-algebraic equations. The system has multiple modes of operations corresponding to discrete states such as "cruising" or "failing". We capture these type of systems as Stochastic Hybrid Systems (SHS).

1.1 System description

We model the thermal management system of a prototypical aircraft executing a certain class of missions. The objective of the study is to determine the fuel temperature distribution over time from take-off to landing. The temperature distribution is not only affected by the type of mission, but it is also sensitive to weather conditions, and these two elements of the system are both uncertain. We classify uncertainty into two categories: parametric uncertainty and dynamic uncertainty. An example of dynamic uncertainty is the mission profile which determines the inputs to the aircraft sub-systems. These inputs, such as commands sent to actuators, are stochastic processes. Parametric uncertainty is associated with the value of a variable in the system that does not change over time. Such value does not change either because it is indeed a constant, or because its dynamics is very slow compare to the time scale of interest. We consider the outside temperature to be an uncertain parameter rather than a stochastic process because we assume that the temperature will not change over the time span of the mission.

Figure 1.1 shows the high level model of the system under study. The mission profile drives the dynamics of all the aircraft sub-systems (the electric power system (EPS), the thermal management system (TMS), the environmental control system (ECS), the engine, and the flight control system). For example, the mission profile determines the power requirements at each point of the mission which in turn determines the heat dissipated by the EPS. Many other variables are directly derived from the mission requirements such as fuel consumption and heat dissipated by the ECS and by the engine. The weather condition impacts the ability to reject heat and therefore has an impact on the dynamics of the fuel temperature.

Figure 1.2 shows the dependency of some of the critical variables of the system. The mission

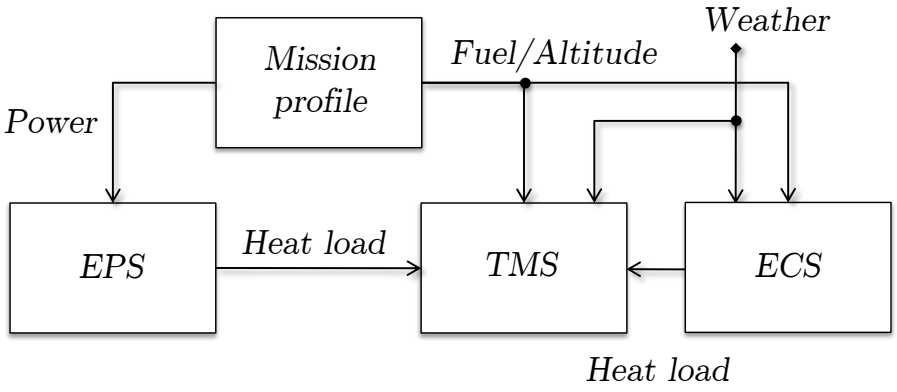


Figure 1.1: High-level model of the system under study.

profile generates a certain power requirement profile $w(t)$, thrust profile $f(t)$, velocity profile $v(t)$, and altitude profile $h(t)$. The mission profile can be modeled at different abstraction levels. At the lowest abstraction level, the details of each component are included in the system, such as detailed models of the dynamics of the aircraft. However, a system level design activity necessarily requires to abstract from low level details and use models that are accurate enough only to make high-level decisions. For example, the dynamics of the aircraft can be abstracted into a noisy velocity variable, while the (x, y) position of the aircraft might not be relevant. The altitude must instead be considered because it has a direct impact on the outside air temperature and, therefore, on the ability to reject heat using ram air. The uncertainty introduced in the model can also be seen as the result of the abstraction. For example, the uncertainty introduced on the velocity variable is not meant to indicate noisy measurements, but rather the speed variation due to several factors such as small maneuvers, turbulence etc. Notice that this variations may also be irrelevant in determining fuel temperate and therefore one may even abstract such uncertainty and consider the aircraft traveling with constant speed in each mission phase.

The EPS system provides power to the actuators and to other sub-systems in the aircraft such as radars and avionics. Power is delivered by the generators which are characterized by a certain efficiency η_G. The heat generated by the EPS depends on the efficiency of the generators and the efficiency η_C of other components (such as power converters) that are present in the aircraft[1].

The altitude and the velocity of the aircraft impact the air temperature T_A and air density d_A. This is important because air is used to extract heat from the fuel. The weather condition also has an impact on T_A as well as on the heat that needs to be rejected by the TMS (due to kinetic effects).

The fuel temperature in the tank depends on the amount of the fuel level which is directly proportional to the thermal capacity of the fuel system. The amount of heat that can be extracted from the fuel system and transferred to the environment using ram air depends on the air density and on the velocity of the aircraft. Finally, the amount of fuel consumption impacts the fuel flow rate in the fuel system and therefore in the heat exchangers, thereby affecting the temperature of the fuel. This is because the temperature of the return fuel is higher for lower fuel flow rates.

1.2 Mission profile

The first component that we model is the mission profile which serves as the context for the rest of the system. Several standard profiles of missions are available in literature [42], ranging from short to long missions which include refueling. We select a standard mission which does not include any

[1]We have omitted other heat loads such as the engine in Figure 1.1 for brevity.

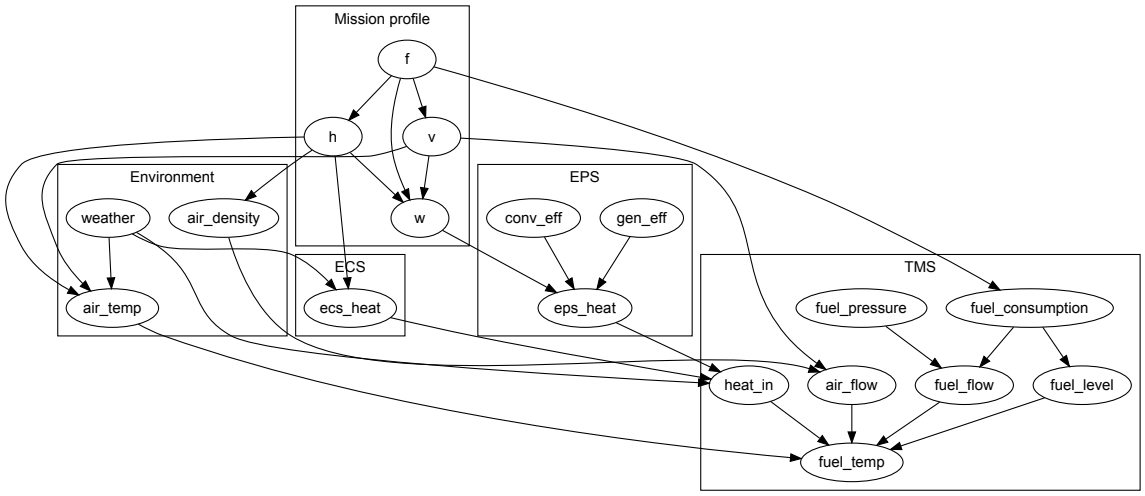


Figure 1.2: Dependencies among the variables and parameters of each sub-system.

complex segment. However, we will be able to capture a class of mission by including uncertainty into the model. We also note that the model of computation that we use (i.e. Stochastic Hybrid Systems [36, 19]) is very general and allows to model a wide variety of mission profiles.

We decompose a mission in the following phases or modes of operation.:

- *Taxing*. In this mode, the aircraft is on the ground and ram air cannot be used as heat sink (we assume that there are no fans on the aircraft). The amount of time spent on the ground is not known at design time and must be modeled by a random parameter.
- *Take-off*. The aircraft moves to this mode and starts accelerating on the ground until enough speed is gained to start the ascending phase.
- *Ascending* to a target altitude. The aircraft climbs at a constant rate until a target altitude is reached. The altitude can be assumed a random variable. However, there is little difference in terms of air temperature and density for a wide range of altitudes as can be seen later in Section 1.3.
- *Flying* at constant altitude. After the target altitude is reached, the aircraft starts flying at that altitude for the duration of the mission which is also assumed to be a random variables. In this mode, the power requirement, and the thrust might be considered random processes simply because the exact maneuvers that the aircraft will do are not known.
- Optional refueling for longer mission that can be executed a certain umber of times:
  - *Descending* to intermediate altitude for refueling. The aircraft descends to an intermediate altitude to fly in formation with a tanker.
  - *Refueling*. This is a mode where the flying conditions are very stable with little uncertainty. The tanker and the aircraft fly in formation maintaining a constant speed and altitude.
  - *Ascending* to target altitude. The aircraft climbs back to the target altitude to continue the mission.
- *Descending*. After complete the mission, the aircraft starts descending towards the landing zone. This phase ends when an appropriate altitude is reached.

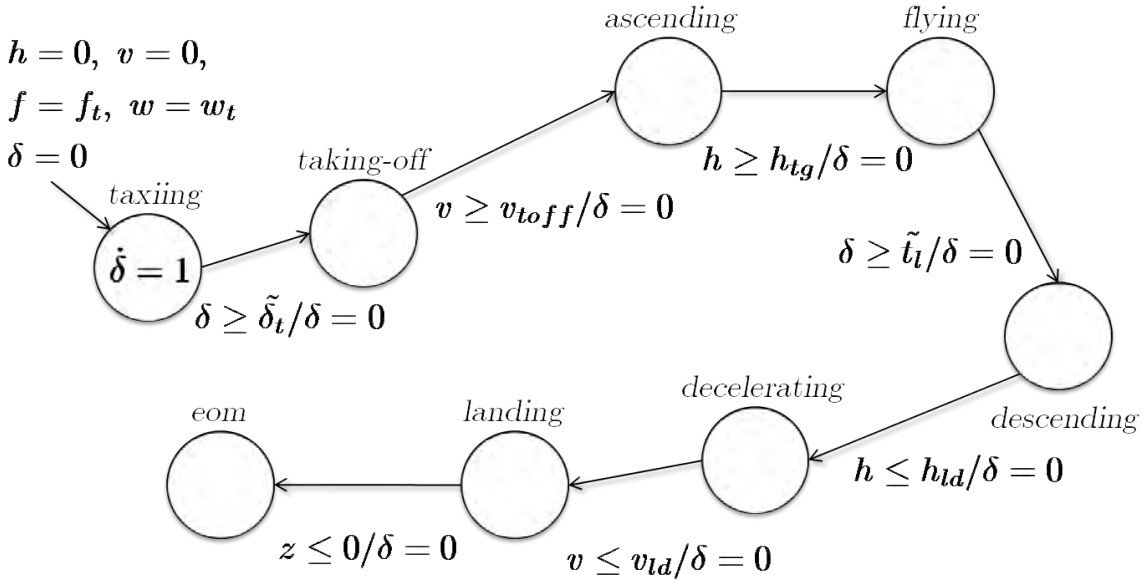


Figure 1.3: Mission profile of a mission without refueling.

- *Decelerating.* The aircraft reduces its speed at a constant altitude. The speed is decreased to a value considered appropriate for landing.
- *Landing.* This is the landing phase where speed is finally reduced to zero.
- *End of mission.* The end of mission corresponds to the engines being shut off.

Figure 1.3 shows the stochastic hybrid system representing a simple mission without refueling. Each state represents one phase of the mission and it is characterized by a set of differential algebraic equations that defines the way in which altitude h, velocity v, thrust f and power requirement w change over time. Variables δ is used to encode time. It is a special case of a class of variables that are referred to as clocks. Clocks have constant derivative, and time is a special case where the derivative is equal to one. To avoid cluttering the figure, we only show the transitions among the states and the dynamics of the clock δ, while we omit the dynamics of the other variables – that are explained later in this section. The aircraft remains on the ground for a certain amount of time $\tilde{\delta}_t$ which is a random parameter. During taxing, the altitude is at the sea level, and the velocity is approximately zero. The aircraft requires some level of thrust f_t and some level of power w_t that are both considered constant.

The take-off phase is characterized by a constant altitude, a constant acceleration of the vehicle and constant thrust and power requirement. When the aircraft reaches a velocity v_{toff}, the mission switches to a phase where the aircraft starts ascending. When a target altitude h_{tg} is reached, the mission changes phase to a state where the aircraft flies at constant altitude. During this phase thrust, power requirement and velocity change according to a set of stochastic differential equations. After a random amount of time $\tilde{t}_l$, the aircraft starts its descending phase and switches to the deceleration phase when the altitude reaches a landing altitude h_{ld}. Finally, the aircraft lands and ends the mission when the ground is touched. The end-of-mission (*eom*) state is an accepting state of this automaton.

The maximum total take-off weight of the aircraft is approximately $50,000\ kg$. We consider that the aircraft is initially full of fuel. The dry mass of the aircraft is $10,400\ kg$, and the total fuel

Mission phase	**Heat load** (kW)	**Power requirement** (kW)
Taxiing	18.44	84.1
Take-off/Decelerating/ Landing	26.63	84.4
Ascending/Descending	27	83
Flying	20	76.4

Table 1.1: Heat load and power requirements for a prototypical UAV.

capacity is $8,400\ kg$. The heat the power requirement in the different phases of the missions are shown in Table 1.1.

The take-off thrust is $17,300\ kg$. We can now define all the parameters in the mission automaton as follows:

- $Taxiing$: even if the aircraft is idling, a substantial thrust is required to keep the engine running. The thrust is needed to keep compressing air into the engine. The amount of thrust is assume to be 25 % of the take-off thrust. Thus, the thrust requirements in this phase is $f_t = 4,325\ kg$ and the power requirement is $w_t = 84.1\ kW$. We assume that the time spent in this mode of operation is uniformly distributed between 5 and 15 minutes.
- $Take-off$: the take-off thrust is $f_{toff} = 17,300\ kg$, i.e. full thrust (we are not considering after-burning). The power requirements is $w_{toff} = 84.4\ kW$. Velocity increases according to the following equation: $\dot{v} = \frac{f - f_{drag}}{m}$ with $f_{drag} = c_d \cdot \rho \cdot A \cdot v^2$, where the drag coefficient $c_d = 0.048$, ρ is the air density and A is the wing area which we consider to be $28\ m^2$. The take-off velocity is $v_{toff} = 240\ km/hr$. Whether this parameters will all be used depends on how they are going to affect the fuel temperature.
- $Ascending$: we assume that the thrust requirement is equal to the take-off thrust. The velocity equation is similar to equation (**??**) but we need to consider an additional term due to gravity $f_g = -g \cdot sin(\pi/2 - \alpha)$ where α is the climb angle which we assume to be $\pi/3$. Also, the air density is going to be a function of the altitude. We assume that the target altitude is $h_{tg} = 10,000\ m$. The total power requirement during climb is $83\ kW$.
- $Flying$: several models are possible for this phase. The simplest model considers velocity and altitude to be constant. A more refined model consists in a system of stochastic differential equations that captures the uncertainty in the types of maneuvers performed in this mission phase. A good approximation is to break this phase into several sub-modes each capturing a particular maneuver. For an example of this type of model, one may refer to the maneuver automat approach [50].
- $Descending$: we assume a descend rate of $h_d = 300\ m/s$, and thrust $f_d = f_t$. The total power requirement is $83\ kW$. The landing altitude is $h_{ld} = 200\ m$.
- $Decelerating$: the aircraft decelerates to a landing speed equal to the take-off speed. During this phase the altitude of the aircraft is maintained constant at h_{ld}.
- $Landing$: The total power requirement is $83\ kW$.
- $End-of-mission$: In this mode all the variables are set to zero.

1.3 Modeling the weather conditions

The weather conditions can be modeled by an uncertain system of algebraic equations linking the altitude to the air density and to the temperature which is also affected by the speed of the aircraft.

Layer	Base Altitude, h_n (km)	Lapse Rate, λ_n (K/km)
0	0	-6.5
1	11	0
2	20	$+1.0$
3	32	$+2.8$
4	47	0
5	51	-2.8
6	71	-2.0
7	84.85	$-$

Table 1.2: Description of various layers in the weather model.

This models is used to compute the actual temperature of the ram air on the heat exchangers. Depending on the weather condition (which is a random parameter), the air density and temperature have a different distribution. These distributions can be computed by fitting historical data.

To model the air density and temperature as functions of altitude, we use models similar to the U.S. Standard Atmosphere models [2]. These models provide definitions for atmospheric temperature, density and pressure over a wide range of altitudes. For these models, the gas which comprises the atmosphere is assumed to be an ideal gas. The key variables of interest to us are:

- h = Altitude
- T = Temperature
- p = Pressure
- g = Acceleration due to gravity = $9.8m/sec^2$
- R = radius of earth = $6356.766km$.
- T_0 = sea level temperature = 15.0^0C
- p_0 = sea level pressure = $101,325N/m^2$.

The model comprises a series of six layers, each defined by a linear temperature gradient also called *lapse rate*. A brief description of the layers is given in table 1.2.

A positive lapse rate $\lambda_n > 0$ indicates that the temperature increases with height. The temperature distribution within layer n is given by:

$$T = T_n + (h - h_n)\lambda_n. \tag{1.1}$$

Using the ideal gas law equations and the equations for hydrostatic equilibrium, the pressure distribution within layer n is given by the expression

$$\frac{p}{p_n} = \left(1 + \frac{(h - h_n)\lambda_n}{T_n}\right)^{-g/(\lambda_n R)}. \tag{1.2}$$

For isothermal layers ($\lambda_n = 0$), the above expression reduces to

$$\frac{p}{p_n} = e^{-(h-h_n)g/(RT_n)}. \tag{1.3}$$

The above equations link the altitude to the pressure and temperature of the air that is used for the fuel-air heat exchangers on the aircraft.

Figure 1.4: Model of the thermal Management System.

1.4 Modeling the Thermal Management System

The first abstract model we consider is shown in Figure 1.4. In this first model we consider the flow rates at steady state or slowly varying. Also, the volume of the fluid in this circuit changes relatively slowly. If this assumption does not hold, then it is possible to better approximate the behavior of the system by using a series of models at constant flow rate, and capturing the transient effect in the transitions among modes.

A pump is used to push fuel from the fuel tank into the fuel circuit. The heat produced by the environmental control system, the electric power system and the engine is absorbed by the fuel through heat exchangers that we abstract in this high level model (see Section 6.1 for a refined model of this system). Part of the fuel is used by the engine while the rest returns to tank. Before entering the tank, the fuel is cooled to an appropriate temperature by an Air/Fuel heat exchanger that uses RAM air. The variables of interest in this model are the total fuel level in the tank, the fuel flow rate, and the fuel temperature.

Using the nomenclature in Figure 1.4, we model the system using the following equations:

- $\dot{M} = \dot{m_{in}} - \dot{m_{out}} = -\dot{m}_f$, $M(0) = M_0$. This equation links the engine fuel consumption $\dot{m}_f$ to the total fuel mass M (where M_0 is the total fuel mass at the beginning of the mission).
- $\dot{m_{out}} c_f (T_f - T_{out}) = H_L$ where H_L is the total heat rate from the heat loads on the aircraft. In this equation c_f is the specific heat of the fuel.
- $\dot{m_{in}} c_f (T_f - T_{in}) = H_S$ where H_S is the heat rate that the sink is able to reject.
- The last equation links the fuel temperature to all the other quantities. Notice that the fuel temperature is T_{out}:
$$\dot{m_{in}} c_f T_{in} - \dot{m_{out}} c_f T_{out} = \frac{d}{dt}(M c_f T)$$
where $T(0) = T_0$ is the initial temperature of the fuel. The expression in explicit form is as follows:
$$\dot{m_{in}} c_f T_{in} - \dot{m_{out}} c_f T_{out} = \dot{M} c_f T + M c_p \dot{T}$$

The heat rate H_S depends on the velocity of the aircraft, the air density and the air temperature.

The fuel rate $\dot{m}_f$ can be derived from the thrust and from the thrust specific fuel consumption (TSFC) of the engine. For example, one representative engine consumes 0.7 $lb/lbt/hr$ (pounds per pounds of thrust per hour) without afterburner, and 2 $lb/lbt/hr$ with afterburner.

Chapter 2

Analysis Techniques for Discrete Time Stochastic Hybrid Systems

The hybrid system model of computation combines the continuous time (CT) and the discrete time (DT) models. A system is characterized by a *discrete* state ranging over a discrete set of values (or modes), and by vectors of continuous states (one vector for each mode). In each mode, the continuous states evolve according to the solution of a system of differential equations (which can be different for each mode). A hybrid system can change mode depending on the value of the continuous states (also called *guard* condition). When changing mode, say from m_1 to m_2, the initial state for the set of differential equations in m_2 is determined by a *reset* relation between the values of the continuous variables in m_1 and in m_2. The guard condition and the reset together define the *jump* condition. For a review of hybrid system models and tools refer to [20]. This formalism is powerful and general and it is able to capture systems where digital controllers interact with physical systems.

Given a hybrid system and a set of possible initial condition for its states (both discrete and continuous), an interesting problem is *reachability analysis* which entails computing the set of all reachable states according to the jump conditions and to the dynamics in each mode. This problem is decidable for *linear hybrid automata* (see [8, 35]). HyTech [35] is a verification tool for linear hybrid automata where the reachable set of states is made finite by using a polyhedral representation. For nonlinear hybrid automata the problem is not decidable. However, it is possible to find a conservative approximation of the original hybrid system as a linear hybrid automata (see for example [28]).

Hybrid systems can be extended to include uncertainty. Each mode can be associated to a system of *stochastic* differential equations capturing the effect of *noisy* dynamics. Jump conditions can be extended by including the probability of switching from one mode to another as function of the state variables. Also, the reset condition can be extended to a probability distribution over the initial conditions of the system of stochastic differential equation in the target mode. The probabilistic reachability analysis problem [7] asks for the computation of the probability distribution over the hybrid state space at a given time starting from an initial probability distribution at time zero. Techniques to solve this problem can then be used to compute the probability of entering into an unsafe set of states.

Computational techniques for solving the probabilistic reachability analysis problem can be divided into two categories: the ones based on a Markovian approximation of the stochastic hybrid system [6, 49, 48], and the ones bases on simulation (or statistical methods) [57, 16]. Another interesting approach is based on Lyapunov-like arguments and is known as the barrier certificate method [47].

In this chapter we first define the type of stochastic hybrid system that we will use in the rest of the study and we will then present the algorithmic implementation of some of the techniques

mentioned in this introduction.

2.1 Markov modeling of Stochastic Hybrid Systems

One possible approach for analysis of stochastic hybrid systems is to use finite state Markov chains. Such techniques have been exploited in the work of Dellnitz ([25]) and Froyland [29] that have used set-oriented methods to model continuous dynamical systems. The basic idea in these methods is to partition the domain of the continuous variables (referred to in the sequel as the continuous state space) into a finite number of sets. To construct the finite state Markov model, the index of each set is identified as the state of the Markov chain. Transition probabilities between different states of the Markov chain are interpreted as the probability of a typical point in one set to move to another set under the constraint on the dynamics of the system. We use the same approach here, except that we take into account the hybrid nature of the dynamics, namely the jump conditions and the different dynamics for each mode of the hybrid system.

Other approximation techniques have been also recently developed by Abate *et al.* [5, 26] and are based on Markov Set-Chains [33]. This work focuses on providing error bounds between the original system and the Markov Chain approximation but places some restrictions on the smoothness of the probability distributions. In our first attempt to address the probabilistic reachability analysis problem, *we do not place restrictions either on the dynamics of the system or on the probability distributions of the stochastic processes.* This complicates the analysis task and does not allow to compute exact error bounds[1].

The rest of this Section is structured as follows. In Section 2.1.1, we provide a definition for Discrete Time Stochastic Hybrid Systems. In Section 2.1.2, we define certain transfer operators that describe the evolution of probability distributions on the hybrid state space. In Section 2.1.4, we discuss numerical methods for approximating the transfer operators. In 2.1.5, we discuss two examples of DTSHS for which we use our computational methods.

2.1.1 Definition of Discrete Time Stochastic Hybrid Systems

We first define state space models for discrete time stochastic hybrid systems (DTSHS). We represent the discrete state space by $\mathcal{Q}$. To keep the explanation of the main idea simple, we consider a continuous state in $\mathbb{R}^n$. In Section 2.3.1 we will remove this assumption and in Section 4.1 we will further extend the model to encompass inputs, outputs and hierarchy. The state space of the hybrid system can then be defined as follows:

$$\mathcal{S} = \mathcal{Q} \times \mathbb{R}^n = \cup_i \{q_i\} \times \mathbb{R}^n. \tag{2.1}$$

The following definition formalizes the state space description of a discrete time stochastic hybrid system.

Definition 1. *The state space model for a discrete time stochastic hybrid system is a collection* $\mathcal{H} = (\mathcal{Q}, Init, T, L, R)$ *where*

- *(***modes***)* $\mathcal{Q} := \{q_1, q_2, ..., q_m\}$ *with* $m \in \mathbb{N}$*, represents the discrete state space;*
- *(***Initial uncertainty***)* $Init : \mathbb{B}(\mathcal{S}) \to [0, 1]$ *is an initial probability measure on* $\mathcal{S}$*.*
- *(***Flows***)* T *is a stochastic map that describes the dynamics of the continuous variables* $x \in \mathbb{R}^n$ *in each mode. The dynamics of the continuous variables in mode* q_i *is given as*

$$x(n+1) = T(q_i, x(n), \xi_i(n)), \tag{2.2}$$

[1] However, the procedure we follow allows to compute error variance empirically.

where $\xi_i(n)$ is an i.i.d process with distribution $\mathcal{N}_i$.

- *(***Switching function***) S is a switching probability function that gives the probability of switching between various modes. $S(x, q_i, .)$ is a probability measure on the discrete space space $\mathcal{Q}$. i.e., $S(x, q_i, q_j)$ gives the probability of the system to jump from mode q_i to mode q_j given the value of the continuous state x.*

- *(***Reset Maps***) R is a stochastic map that probabilistically resets the values of the continuous state variables when a switch occurs from mode q_i to mode q_j. The reset is given as*

$$x(n+1) = R(q_i, q_j, x(n), \eta_j(n)), \tag{2.3}$$

where $\eta_j(n)$ is an i.i.d process with distribution $\mathcal{W}_j$.

The execution of a state space model for the discrete time stochastic hybrid system over a finite time horizon $[0, N]$ is defined below [5].

Definition 2. *Consider the state space model for a DTSHS $\mathcal{H} = (\mathcal{Q}, Init, T, L, R)$. An execution of the model over a time horizon $[0, N]$ is given by the following algorithm:*

Set $k = 0$ and extract a value $(q(0), x(0))$ according to the distribution Init
while *$k < N$* ***do***
  Extract a value $q(k+1)$ according to the probabiity distribution $S(x(k), q(k), .)$.
  if *$q(k+1) = q(k)$* ***then***
    Extract a value $\xi_i(k)$ according to the distribution $\mathcal{N}_i$. Then compute

$$x(k+1) = T(q(k), x(k), \xi_i(k)), \tag{2.4}$$

    {i is the index of the mode $q(k)$.}
  else
    Extract a value $\eta_j(k)$ according to the distribution $\mathcal{W}_j$. Then compute

$$x(k+1) = R(q(k), q(k+1), x(k), \eta_j(k)), \tag{2.5}$$

    {j is the index of the mode $q(k+1)$.}
  end if
  $k \to k+1$
end while

2.1.2 Propagation of measures for Discrete Time Stochastic Hybrid Systems

The initial probability measure of a DTSHS needs to be propagated over time according to its hybrid dynamics. To propagate the probability measure we define transfer operators for the flow maps and for the jumps. With a slight abuse of notation, we use the same symbols for measures and probability distribution functions.

Definition 3. Flow Transfer Operators: *For the flow corresponding to each mode q_i, the propagation of measures is described by the Frobenius-Perron operator corresponding to the map $T(q_i, ., .)$. This is the unique operator $[P_i]$ such that*

$$\int_A [P_i]\mu(x)dx = \mathbb{E}_{\xi_i}\left[\int_{\mathbb{R}^n} \mu(x).\chi_A(T(q_i, x, \xi_i))dx\right], \tag{2.6}$$
$$\text{for all } A \subset \mathbb{R}^n.$$

Note that if the probability measure at time k is given by $\mu(k)$, then the measure at time $k+1$ is given as

$$\mu(k+1) = [P_i]\mu(k). \tag{2.7}$$

Note that eventhough the maps $T(q(k), x(k), \xi_i(k))$ are non-linear, the Frobenius-Perron operators are linear operators, but infinite-dimensional. For more details on the theory of these transfer operators, see [41].

Definition 4. Switching Transfer Operator: *For the switching between the modes q_i and q_j, we define the switching transfer operator given as*

$$[L_{i,j}]\mu(x) = S(x, q_i, q_j).\mu(x). \tag{2.8}$$

Definition 5. Reset Transfer Operators *The change in measures due to probabilistic resets is given by the Frobenius-Perron operator corresponding to the map $R(q_i, q_j, ., .)$. This is given as*

$$\int_A [M_{i,_j}]\mu(x)dx = \mathbb{E}_{\eta_j}\left[\int_{\mathbb{R}^n} \mu(x).\chi_A(R(q_i, q_j, x, \eta_j))dx\right], \quad \text{for all } A \subset \mathbb{R}^n. \tag{2.9}$$

Let Γ be a probability distribution on the state space of the hybrid system $\mathcal{S}$. We define the following sub-probability measures on the continuous space $\mathbb{R}^n$.

$$\mu_i(A) = \Gamma(q_i, A), \text{ for } i = 1, 2, ...m. \tag{2.10}$$

Note that $\mu_i(\mathbb{R}^n) \leq 1$. Hence it is a sub-probability measure. Also note that the probability of the state being in the set $A \subset \mathbb{R}^n$ (irrespective of the mode), is given by

$$\mu(A) = \sum_{i=1}^{m} \mu_i(A). \tag{2.11}$$

The evolution of the probability measure over the state $\mathcal{S}$ of the $DTSHS$ $\mathcal{H}$ over a finite time-horizon $[0, N]$ is given by the following algorithm:

Definition 6. Algorithm for Propagation of measures

Set $k = 0$ and set $\Gamma^0 = Init$.
while *$k < N$* ***do***
  for *$i = 1, 2, ..., m$* ***do***
    Get the sub-probability measures.

$$\mu_i^k(.) = \Gamma^k(q_i, .) \tag{2.12}$$

  end for
  for *$i = 1, 2, ..., m$* ***do***
    for *$j = 1, 2, ..., m$* ***do***
      Compute the sub-probability measures

$$[\rho_{i,j}^k]^- = [L_{i,j}]\mu_i^k \tag{2.13}$$

      Reset the sub-probability measures $[\rho_{i,j}^k]^-$

$$\rho_{i,j}^k = [M_{i,j}][\rho_{i,j}^k]^-. \tag{2.14}$$

end for
end for
for $i = 1, 2, ..., m$ ***do***
Compute the sub-probability measures $[\mu_i^{k+1}]^-$.

$$[\mu_i^{k+1}]^- = \mu_i^k(.) - \sum_{j=1}^{m} [\rho_{i,j}^k]^-(.) + \sum_{j=1}^{m} \rho_{j,i}^k(.) \tag{2.15}$$

end for
for $i = 1, 2, ..., m$ ***do***
Compute (evolve sub-probability measures for one time-step by local flow)

$$\mu_i^{k+1} = [P_i]\,[\mu_i^{k+1}]^- \tag{2.16}$$

Set $\Gamma^{k+1}(q_i, .) = \mu_i^{k+1}(.)$.
end for
$k \to k+1$
end while

Description of the algorithm: The first step is to compute the probability "mass" that exits from mode q_i to mode q_j represented by the sub-probability measures $[\rho_{i,j}^k]^-$ given in (2.13). The sub-probability measures $[\rho_{i,j}^k]^-$ need to be updated according to the reset map R. This is described in Equation (2.14). Once this is done, we compute the sub-probability measures $[\mu_i^{k+1}]^-$ as obtained by Equation (2.15). This step essentially substracts the mass that exited from mode i to other modes and adds the mass that came in from other modes (after the reset). The next step is to propagate the sub-probability measures $[\mu_i^{k+1}]^-$ according to the dynamics of the mode q_i. This is given by Equation (2.16).

2.1.3 Transfer operator for the hybrid state space

The evolution of the probability measures over the whole hybrid state space can be described using a single transfer operator given as

$$\Gamma^{k+1} = \mathbb{P}\Gamma^k, \tag{2.17}$$

where

$$\Gamma^k = \begin{bmatrix} \mu_1^k \\ \mu_2^k \\ .... \\ .... \\ \mu_m^k \end{bmatrix}, \tag{2.18}$$

and the transfer operator $\mathbb{P}$ can be represented in block-operator form as

$$\mathbb{P} = \begin{bmatrix} P_1 M_{1,1} L_{1,1} & P_2 M_{2,1} L_{2,1} & ..... & P_m M_{m,1} L_{m,1} \\ P_1 M_{1,2} L_{1,2} & P_2 M_{2,2} L_{2,2} & ..... & P_m M_{m,2} L_{m,2} \\ ..... & ..... & ..... & ..... \\ ..... & ..... & ..... & ..... \\ P_1 M_{1,m} L_{1,m} & P_2 M_{2,m} L_{2,m} & ..... & P_m M_{m,m} L_{m,m} \end{bmatrix}. \tag{2.19}$$

Each block of the above operator is a composition of the various flow, switching and reset transfer operators. The operator defined above represents in a more compact form the action for measure propagation as described in the algorithm defined above in the previous section.

2.1.4 Numerical approximation of Transfer Operators for Discrete Time Stochastic Hybrid Systems

In [25], Dellnitz et al. describe set oriented numerical methods to construct finite dimensional approximations for the Frobenius-Perron operator corresponding to a continuous dynamical system. In this paper, we use the same techniques to construct approximations for the various transfer operators defined in the previous section. In this approach, the dynamics is modeled by a finite state Markov chain.

As described in [25], the transfer operator corresponding to a map T is constructed as follows. The state space is partitioned into a finite number of connected sets $\{A_1, A_2, ..., A_n\}$. To form the Markov model, each set A_i is identified with a state i of an n-state Markov chain. A $n \times n$ matrix P is constructed, where the entry P_{ij} is computed as

$$P_{ij} = \frac{m(A_i \cap T^{-1} A_j)}{m(A_i)}. \tag{2.20}$$

where m is the Lebesgue measure. The quantity P_{ij} can be interpreted as the probability that a typical point in A_i moves into A_j under one iteration of the map T. The quantity P_{ij} is computed by a Monte-Carlo approach. One randomly selects a large number of points $\{a_1, ..., a_N\} \subset A_i$ and sets $P_{ij} \approx \#\{a \in \{a_1, ..., a_N\} : T(a) \in A_j\}/N$. We construct such approximations for each one of the Flow Transfer Operators ($[P_i]$) and Reset Transfer Operators ($[M_{i,j}]$). There is no need to explicitly construct the Switching Transfer operator as its action is known exactly in terms of the switching probability function. **Remarks.** The matrix representation of the transfer operator could in be directly used for measure propagation. This requires to compute the quantity P_{ij} for each state i of the finite Markov Chain approximation even if state i is actually never reached by the dynamics of the system. When the number of state of the approximation is small, one may use this method and further improve memory and computational requirements by using a sparse matrix representation. However, constructing Markov models for a given specification of a hybrid system is computationally intensive. To give some idea of the computational complexity, consider a hybrid system with m discrete states and N continuous variables, each sampled using G intervals. Then, the total number of states for the Markov chain approximation is $O(m.G^N)$, and the size of the finite dimensional Frobenius-Perron operators is $O((m.G^N)^2)$. We show in Section 2.3 how the computation of transfer operators can be interleaved with measure propagation to avoid considering those parts of the state space that are never reached. Finally, we point out that the probabilities P_{ij} are average probabilities and that their computation can be simplified when the flow and reset maps have particular forms (see for example Section 2.3.4).

One application of Markov models is to propagate probability measures for the state of the hybrid system as described in the previous sections. Another possible application is to use probabilistic model checkers developed for Markovian models ([44, 38, 1]). Our software tool takes as input the specification of a hybrid system and returns an approximate Markov chain model of the system.

2.1.5 Examples

In this section we present some simple examples of systems for illustrative purposes. This models are meant to provide an overview of the use of DTSHS and how measures propagates through them in the present of jumps.

Thermostat

The temperature dynamics of a thermostat regulated room can be modeled by a hybrid system. In this example, the hybrid system has two modes corresponding to a heater being on and off. The

dynamics of the temperature for these two modes are given as:

$$\mathbf{OFF}(\mathbf{0}) : x(k+1) = x(k) - 0.25 + \xi(k) \tag{2.21}$$

$$\mathbf{ON}(\mathbf{1}) : x(k+1) = x(k) - 0.25 + H + \xi(k) \tag{2.22}$$

where H is heat injected by the heater in the room at each sampling step. The switching probability function for the system is defined as follows:

$$\begin{aligned} S(x,0,1) &= \begin{cases} 0 \text{ if } x > x_{lmax} \\ \frac{x_{lmax}-x}{x_{lmax}-x_{lmin}} \text{ if } x_{lmin} \leq x \leq x_{lmax} \\ 1 \text{ if } x < x_{lmin} \end{cases} \\ S(x,0,0) &= 1 - S(x,0,1) \\ S(x,1,0) &= \begin{cases} 0 \text{ if } x < x_{hmin} \\ \frac{x-x_{hmin}}{x_{hmax}-x_{hmin}} \text{ if } x_{hmin} \leq x \leq x_{hmax} \\ 1 \text{ if } x > x_{hmax} \end{cases} \\ S(x,1,1) &= 1 - S(x,1,0). \end{aligned} \tag{2.23}$$

The reset map for all mode transitions is identity. i.e., there is no change in the continuous state of the system after a mode transition. We set $H = 0.5$, $x_{lmin} = 48.0, x_{lmax} = 52.0, x_{hmin} = 58.0$ and $x_{hmax} = 62.0$. Figure 2.1 shows snapshots of the evolving probability measure at various times. We denote with μ_1 and μ_2 the temperature probability distribution in the **OFF** and **ON** states, respectively. The thermostat is initially in **OFF** and the temperature in the room is uniformly distributed between 60 and 65.

Bouncing Ball

In this example, we consider a ball bouncing on the ground. The system has only one mode, but there is a reset condition when the position of the ball goes below zero. When this even occurs, the value of the velocity is reverse (i.e. the ball start going upwards). The reset map is stochastic to capture effects of the uneven surface of the ball and of the pavement. The dynamics for the only mode of the system is given as:

$$\begin{aligned} x(k+1) &= x(k) + v(k).\Delta \\ v(k+1) &= v(k) - g\Delta \end{aligned} \tag{2.24}$$

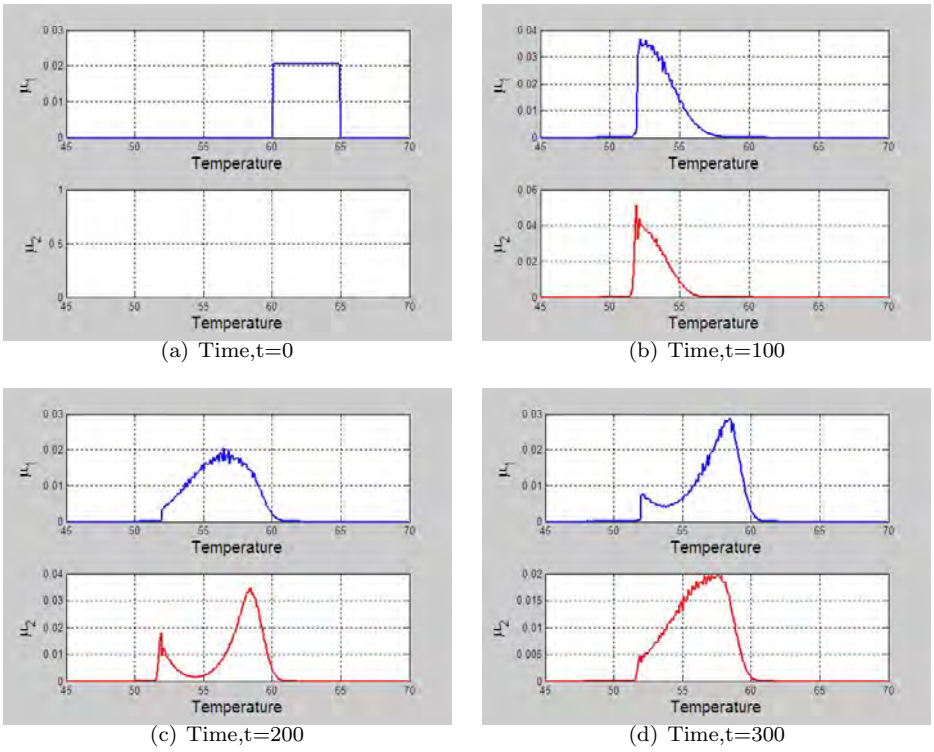
where Δ is the sampling rate. The reset map is identity when the position $x(k)$ is greater than zero. When $x(k)$ is less than zero, the position and velocity are reset as

$$\begin{aligned} x(k+1) &= -x(k) \\ v(k+1) &= -cv(k) + \eta(k). \end{aligned} \tag{2.25}$$

Figure 2.2 shows snapshots of the probability measure evolving according to the bouncing ball dynamics with the stochastic reset. The position of the ball is on the y axis while the velocity is on the x axis. Figure 2.2(c) shows the effect of resetting the velocity from a negative to a positive value.

2.2 Simulation methods: Statistical Model Checking

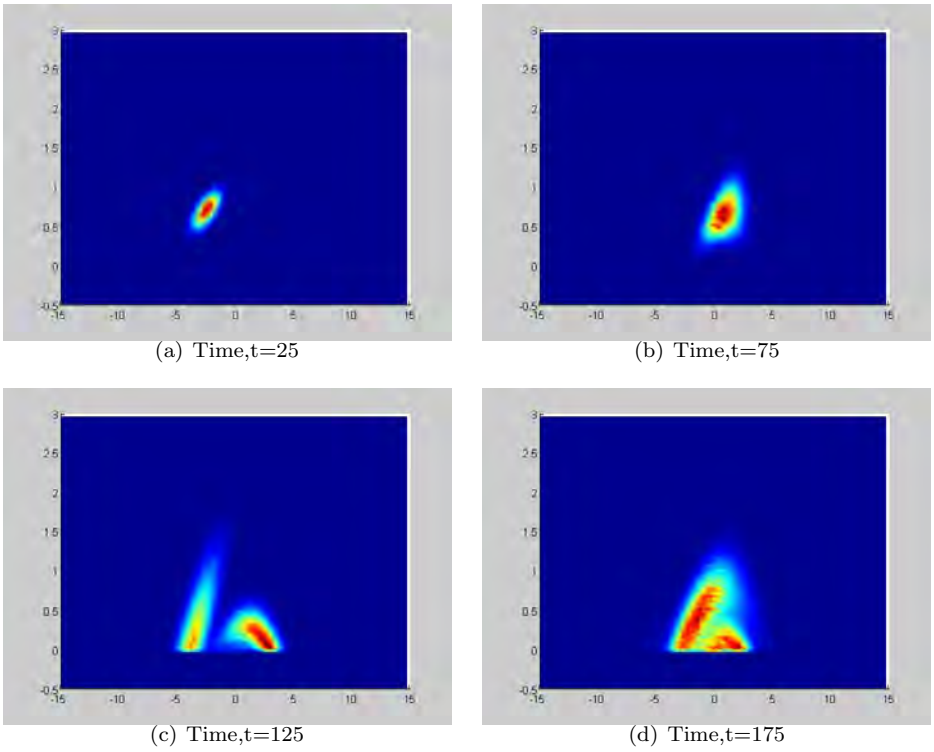
In this section we review some of the simulation-based approaches to the verification of probabilistic systems. These methods estimate the probability that a system satisfies a certain property by



(a) Time,t=0 (b) Time,t=100

(c) Time,t=200 (d) Time,t=300

Figure 2.1: Snapshots at various times of the probability measures for the thermostat example with probabilistic switching between the modes.



(a) Time,t=25 (b) Time,t=75

(c) Time,t=125 (d) Time,t=175

Figure 2.2: Snapshots at various times of the probability measure evolving according to the bouncing ball dynamics with a stochastic reset.

running statistically independent simulations and checking if the resulting traces satisfy the property. The result of checking the property is interpreted as a Bernoulli trial. Let ϕ be a property that can be checked for satisfaction over finite simulation traces[2]. For each simulation of the system, let x_i be a random variable which is equal to 1 if ϕ is satisfied and 0 otherwise. We are interested in testing whether the actual probability p (not known) that the system satisfies property ϕ is above or below a certain threshold θ. This problem can be used using hypothesis testing where the null hypothesis is $H : p \geq \theta$ and the alternative hypothesis is $K : p < \theta$. This problem is relaxed by introducing an indifference region around the actual probability p. Let δ be the length of the indifference region, then new hypothesis are $H' : p \geq p_h = \theta + \delta/2$ and $K' : p < p_k = \theta - \delta/2$.

Consider now a certain number n of samples (i.e. simulations that have been used to check the property) x_1,...,x_n. Checking wether the null hypothesis holds can be done by fixing a number c (which needs to be determined) and checking if $x_1 + ... + x_n \geq c$. Given this simple test, one can compute the probability that the test accepts a false hypothesis. The actual probability that the the sum of the n outcomes is less than or equal to c is:

$$\binom{P_{p,n,c} = \sum_{i=0}^{c}}{nip^i(1-p)^{n-i}}$$

This is the probability of accepting the alternative hypothesis. Thus, the probability of accepting the the null hypothesis is $1 - P_{p,n,c}$. Consider now the following error probability bounds:

- α is the probability of accepting the null hypothesis when the alternative hypothesis holds.
- β is the probability of accepting the alternative hypothesis when the null hypothesis holds.

The pair (α, β) is called the strength of the hypothesis test. To have a test with such strength, the following two constraints must be satisfied:

- $P_{p,n,c} \leq \alpha$ for all $p \geq p_h$, meaning that the probability of accepting the alternative hypothesis when the null holds is below α.
- $1 - P_{p,n,c} \leq \beta$ for all $p \leq p_k$, meaning that the probability of accepting the null hypothesis when the alternative holds is below β.

One can then find n and c that satisfy these two constraints. Ideally, one wants to minimize n (i.e. the number of samples required). The resulting optimization problem is difficult to solve given the non-linearity of the constraints, but approximations can be found. This simple test does not leverage the information gather as the simulations are generated. In fact, if while running the simulations one realizes that the number of times the property was verified is greater than c, then the test could stop without having to generate all n tests.

An different approach is to develop a sequential test that leverages the results obtained by previous checks. This technique is the Sequential Probability Ration Test (SPRT) [51]. The test proceeds as follows. Let m denote the test number (or simulation run number) and let $s_m = \sum_{i=0}^{m} x_i$ be the number of times that the property is found to be true. The one can compute the following ratio:

$$r_m = \frac{p_{k,m}}{p_{h,m}} = \frac{p_k^{s_m}(1-p_k)^{m-s_m}}{p_h^{s_m}(1-p_h)^{m-s_m}}$$

The nominator is the probability of having s_m successes over m trials assuming that the probability of success is p_k, while the denominator is the probability of having s_m successes over m trials assuming that the probability of success is p_h. One than accepts H' if $r_m \leq R_H$ and K' if $r_m \geq R_K$ (with $R_H < R_K$). The problem is to find R_H and R_K so that the test has the required strength which is

[2]Examples of such properties are the ones expressed using Bounded Linear Temporal Logic.

non-trivial. This technique has been used as the basis to develop Statistical Model Checking tools. The method was first introduced by Younes *et al.*[54, 55].

A recent development based on Bayesian Hypothesis Testing is presented in [56]. The conditional probability of x_i given that $p = u$ is:

$$f(x_i|u) = u^{x_i}(1-u)^{1-x_i}. \tag{2.26}$$

A key idea in this approach is that since p is unknown, it can be assumed to be a random variable u whose density $g(.)$ is called the prior density. We can compute the posterior density for the random variable u using Bayes' theorem given as:

$$f(u|x_1, ..., x_n) = \frac{f(x_1, ..., x_n|u)g(u)}{\int_0^1 f(x_1, ..., x_n|v)g(v)dv}. \tag{2.27}$$

Note that since the variables x_i are independent and identically distributed, the probability $f(x_1, ..., x_n|u)$ is given as $\Pi_{i=1}^n f(x_i|u)$. The calculation above is repeated for an unknown number of simulations until the mass of the posterior density within a specified small interval achieves a certain coverage goal. The mean of the posterior density $f(u|x_1, ..., x_n)$ can be used as an estimate of the real probability p. Note that the answer made to the hypothesis testing problem can be incorrect for a finite number of simulation. But it is possible to bound the probability of generating an incorrect answer to the hypothesis testing problem.

Remarks. Statistical approaches are appealing because they only require a simulation model of the system and do not explicitly explore the entire set of reachable states. However, the model cannot exhibit non-determinism which is used in practice to model behaviors which are not fulling known. Some recent work address this problem [17, 4]. It is also important to mention that the outcome of the hypothesis test may be wrong with some probability. Finally, the efficiency of the method (in terms of number of samples required to complete the test) can be improved by using smart sampling techniques such as Quasi Monte Carlo methods.

2.3 Efficient implementation of the reachability algorithm

In this section we present algorithms and data structures for an efficient implementation of the conceptual algorithm presented in Section 2.1. In particular we discuss the discretization of the hybrid state space, the encoding of the grid points, a data structure to store the reachable states and an algorithm to compute the reachability graph and to propagate probability measures.

The algorithm described in Section 2.1 could be implemented simply by computing the transition probabilities between different grid cells for a given discretization, and storing them in a sparse matrix data structure. This transition matrix could then be used to perform matrix-vector multiplications for the evolution of the probability distribution. Note that computing these transition probabilities is an expensive step as we have to uniformly sample points within each cell of the discretization, and compute the action of the flow and reset operators on each one of these points. However, many states of the finite-state Markov chain approximation (or cells of the discretized space) are never visited - hence making it unnecessary to compute the transition probabilities for parts of the state-space corresponding to the cells that are never visited.

This motivated the development of an explicit reachability algorithm that learns the transition probabilities on-the-fly while the initial probability distribution is propagated. The transition probabilities are therefore computed only for those cells that have a non-zero probability of being reached. Although the theoretical complexity of the algorithm does not change, the dynamics of the system constraints the reachable state space considerably making the explicit reachablity analysis capable of handling reasonably sized systems. In the following, we first discuss the encoding of the cells of the partitioned hybrid state space. Then we discuss some data structures that are required for the

efficient implementation of the reachability algorithm. Finally we formally describe the reachability algorithm.

2.3.1 Encoding the state space

Each mode q_i is characterized by an invariant region $Inv(q_i)$ that for our models can be computed as the conjunction of the complement of the guard conditions associated with the switching function from mode q_i to all other modes. The invariant gives information on the bounds of the variables in each mode. In general, the invariant region is a subset of $\mathbb{R}^{d(q_i)}$, where $d(q_i)$ is the dimension of the continuous state space in mode q_i. We limit our discussion to invariant regions that are orthotopes, meaning products of intervals. The invariant orthotope is defined as follows:

$$Inv(q_i) = \left[x_{1,l}^{(i)}, x_{1,u}^{(i)}\right) \times \ldots \times \left[x_{n,l}^{(i)}, x_{n,u}^{(i)}\right) \tag{2.28}$$

where we denoted by $x_{j,l}^{(i)}$ and $x_{j,u}^{(i)}$ the lower and upper bound of the j-th variable in the i-th mode, respectively. To build a finite representation of the invariant region (and therefore of the LSHS) we discretize each continuous state using a grid. For a mode q_i, let $G^{(i)} = (g_1^{(i)}, \ldots, g_n^{(i)})$ represent the grid, where $g_d^{(i)}$ is the number if intervals for the d-th state. The length of each interval is computed as follows:

$$l_d^{(i)} = \frac{x_{d,u}^{(i)} - x_{d,l}^{(i)}}{g_d^{(i)}} \tag{2.29}$$

The j-th interval is defined as follows:

$$I_{d,j}^{(i)} = \left[x_{d,l}^{(i)} + (j-1)l_d^{(i)}, x_{d,l}^{(i)} + jl_d^{(i)}\right) \tag{2.30}$$

The discretization induces a new state space which is now finite and it is defined as:

$$\hat{S} = \bigcup_{i=1}^{m} \{q_i\} \times [1, g_1^{(i)}] \times \ldots \times [1, g_n^{(i)}] \tag{2.31}$$

Let $s \in \hat{S}$ be a state such that $s = (q_i, j_1, \ldots, j_n)$. Then, we use the following notation: $mode(s) = q_i$, $B(s) = (j_1, \ldots, j_n)$ and $H(s) = I_{1,j_1}^{(i)} \times \ldots \times I_{n,j_n}^{(i)}$.

The finite state space is now encoded into a linear array for each mode. For a given mode, let the following define the total number of discrete states in that mode:

$$n_m^{(i)} = \prod_{d=1}^{D(q_i)} g_d^{(i)}$$

The discrete linear hybrid state space is $\hat{S}_L = \cup_{q \in Q}\{q\} \times [1, n_m^{(i)}]$. The encoding is implemented by a function $map : \hat{S} \to \hat{S}_L$ that given a hybrid state $s \in \hat{S}$ computes $s' \in \hat{S}_L$ as follows:

$$\begin{aligned} &mode(s) = mode(s') = q \\ &B(s')|_1 = B(s)|_{D(q)} + \sum_{d=1}^{D(q)} \left(\prod_{d'=d+1}^{D(q)} g_d^{(i)} \right) (B(s)|_d - 1) \end{aligned}$$

This mapping (which is a bijection) has low complexity because it requires a number of operations equal to the dimension of the continuous state space which is generally small. The inverse mapping has the same complexity. By using these two maps, we remove the need for having an explicit grid loaded in memory. Each reachable state will only have an id corresponding to its linear encoding.

2.3.2 Data structures for reachability analysis

The reachability analysis algorithm explores the set of reachable states. We will implement an explicit algorithm which requires the definition of a data structure to maintain the states reached during the execution of the algorithm. The algorithm executes two key operations:

- Given a state s, the set of states S_R reachable from s need to be computed. We provide some approaches to speed up this step later in this chapter.
- Given a new state s', the algorithm needs to check if s' has already been reached.

The basic data structure used to hold the reachable state space is a reachability graph $R_G(C, E, P)$ where C is a set of vertexes, E is a set of edges and $P : E \to [0, 1]$ is a function that associates a probability to each edge. A cell is a tuple $c(s, \mu) \in C$ such that $s \in \hat{S}_L$, $\mu \in [0, 1]^2$ representing the probability of being in state s – the probability measure is a vector of two elements for efficiency reasons as explained in in Section 2.3.3. Because we expect R_G to be sparse, we use an adjacency list representation

To check whether a state has already been reached is a frequent operation which needs to be optimized. There are two methods that we have tested. The first method requires the use of a bit matrix M_V where each bit represents a possible state of the system. Given a state $s \in \hat{S}_L$, $M_V[mode(s)][B(s)]$ is equal to 1 if s has been reached and zero otherwise. Using this auxiliary data structure allows to check whether a cell C has been reached in constant time because the matrix provides direct access to its elements. However, this matrix needs to be allocated for the whole state space which turns out to be the main memory bottleneck. Thus, we decided to store the set of vertexes C in a priority queue that is ordered according to the state s lexicographically. This ordering guarantees $O(\log(|C|))$ access time which is efficient even when the number of reached states is of the order of millions. The memory requirement is only determined by the set of reachable states.

2.3.3 Description of the Algorithm

Algorithm 1: Reachability and measure propagation

Input: DTSHS $\mathcal{H}$, N
Output: $R_G(Q, E, P)$

```
1  k ← 0 ;
   /* Initialize the reachable set                                          */
2  Q ← Init(H) ;
   /* Q_n contains the frontier of new reached cells                        */
3  Q_n[0] ← Q ;
4  while k < N do
       /* i and i_n are the previous and current indices, respectively      */
5      i ← 1 − kmod2, i_n ← kmod2 ;
6      Q_n[i_n] ← ∅ ;
       /* for all new cell reached in the previous iteration               */
7      forall the c(s, μ) ∈ Q_n[i] do
           /* Compute the local PF operator                                */
8          (S', P') ← LocalPf(s) ;
9          forall the s' ∈ S' do
10             c' ← (s', (0, 0)) ;
               /* Add the newly reached cells to the queue if they have not been
                  reached already                                           */
11             if c' ∉ Q then
12                 Q ← Q ∪ {c'} ;
13                 Q_n[i_n] ← Q_n[i_n] ∪ {c'} ;
14             end
15             E ← E ∪ {(c, c')} ;
16             P(c, c') ← P'(s') ;
17         end
18     end
       /* Propagate the probability measure                                */
19     forall the c(s, μ) ∈ Q do
20         μ[i_n] ← 0 ;
21     end
22     s forall the (c(s, μ), c'(s', μ')) ∈ E do
23         μ'[i_n] ← μ'[i_n] + μ[i] · P(c, c');
24     end
25 end
```

Algorithm 1 shows the details of the implementation of the reachability algorithm. At first, the reader may recognize that the algorithm is based on a breath first search over the discrete state space. Although this is the indeed the case, the computational and memory efficiency of the algorithm depend on the implementation details which are hidden in this high level description. Moreover, the graph is constructed during execution which requires to handle queues of newly discovered nodes.

We use pairs of probability measures for each cell and a pair of queues Q_n. By using pairs we avoid possibly expensive copy operations during the update of the queues and, most importantly, of the probability measure. In fact, at each step, the new probability measure needs to be updated according to the old value which would require saving the old value in a cloned data structure. Thus, the update is done by using one element of the pairs in the odd iterations and the other element in the even iterations.

The algorithm starts by initializing the queue of reached cells Q and the current queue of states

to be processed $Q_n[1]$ (i.e. new states found in the previous iteration) to the set of initial states with a non-zero probability. This is done by function `Init` which computes the set of states $S_0 \subseteq \hat{S}_L$ using the $Init$ function of the DTSHS $\mathcal{H}$.

Two indexes are generated at each iteration. The indexes of the queue containing the cells found in the previous iteration i, and the index of the queue which will contain all the new states reachable in one step i_n. For each cell in $Q_n[i]$, a function `LocalPf` computes a set of new states $S' \in \hat{S}_L$ with associated probabilities P' (i.e. the probability of reaching state $s' \in S'$ from s). Each new state is added to the next queue $Q_n[i_n]$ and to queue Q. Finally, the edges of the graph and the associated probability are also added to the reachability graph.

The second step in the algorithm propagates the probability measures. For each cell in the reachability graph, probability mass is moved to the output cells according to the previous value of the probability measure and to the transition probabilities P.

2.3.4 Further improvements

Dealing with clocks It is usual that hybrid system models have mode transitions that are enabled by guards conditions on clock variables. Clock variables can be used to keep track of the amount of time spend in a mode or simply to model continuous states that evolve according to a constant derivative. The dynamics of clock variables are simple ($\dot{\delta} = c$) and usually the dynamics of the other continuous state variables are independent of the clock value. This corresponds to the notion of *time-invariant* dynamical systems. We consider in this section $c = 1$ but the technique can be applied to any other value.

Let $s \in \hat{S}$ be a new discrete state such that $s = (q_i, \delta, j_1, \ldots, j_n)$ where δ is a clock variable. A new translated state $s_t = (q_i, 0, j_1, \ldots, j_n)$ is created by setting the clock variables of the original state s to zero. If s_t has been visited, we compute the discrete states to which s_t has transitions and the transition probabilities by uniformly sampling the cell corresponding to s_t. The states to which s_t has transitions to have the form $s_k = (q_i, 1, k_1, \ldots, k_n)$ because the clock variable is always incremented by one. Once the transitions of s_t are known, we also know the transitions of s. i.e, if s_t has a transition to s_k,then s has a transition to $s_{kt} = (q_i, \delta+1, k_1, \ldots, k_n)$ with the same probability. This avoids having to sample the cell corresponding to the state s. For any state s that is newly visited, if its corresponding translated state s_t has already been visited, then we can just use the list of states to which s_t has transitions, to get the list of states to which s has transitions. Note that this technique can be used only if no mode transitions are enabled by the state s.

Removing elements from the queue During the execution of the reachability algorithm, the probability associated to many cells that are in the queue of reached states could eventually become zero. Some of these states may never be visited again. This is particularly true in systems with clock variables for states that have non-zero values for the clock. This is because all the probability mass in the state $s = (q_i, \delta, j_1, \ldots, j_n)$ are transferred to states of the form $s_k = (q_i, \delta+1, k_1, \ldots, k_n)$ thus leaving no probability mass in the state s. Such states can be removed from the queue thereby reducing memory requirements. During the execution of the reachability algorithm we periodically traverse the queue Q of reached states and we remove states that have non-zero values for the clock and zero probability mass. Notice that there is a trade-off between the interval at which this operation is done and the memory requirement. Traversing the queue of reached states is an expensive operation but can reduce memory requirements. The removal is not a trivial process and it is implemented over multiple passes over the Q starting from those cells which do not have input edges in the reachability graph.

Linear Stochastic Hybrid Systems with additive noise Consider the case where the following restrictions are imposed on the dynamics of the DTSHS:

- (**Flows**)T is a stochastic map that describes the dynamics of the continuous variables corresponding to each mode. The dynamics of the continuous variables corresponding to mode q_i is given as
$$x(k+1) = T(q_i, x(k), \xi_i(k)) = x(k) + \tilde{\xi}_i(k), \tag{2.32}$$
where $\tilde{\xi}_i(k)$ is an i.i.d process characterized by a joint probability distribution function $F_{\tilde{\xi}_i}(\xi_i) = P(\tilde{\xi}_i \leq \xi_i)$.

- (**Reset Maps**) R is a stochastic map that probabilistically resets the values of the continuous state variables when a switch is made from mode q_i to mode q_j. The reset is given as
$$x(k+1) = R(q_i, q_j, x(k), \tilde{\eta}_{ij}(k)) = x(k) + \tilde{\eta}_{ij}(k), \tag{2.33}$$
where $\tilde{\eta}_{ij}(k)$ is an i.i.d process characterized by a joint probability distribution function $F_{\tilde{\eta}_{ij}}(\eta_{ij}) = P(\tilde{\eta_{ij}} \leq \eta_{ij})$.

Let $L^{(i)} = diag(l_1^{(i)}, \ldots, l_n^{(i)})$ be the diagonal matrix with entries equal to the interval length for each dimension. We seek a matrix $P \in \mathbb{R}^{|\hat{S}|\times|\hat{S}|}$ where the entry $P(s, s')$ is the probability of switching from $s \in \hat{S}$ to $s' \in \hat{S}$. Notice that this is an average probability since different points in $H(s)$ will have different probabilities to move to $H(s')$ in one step. Consider a state value $x(k) \in H(s)$ at time k such that $mode(s) = mode(s') = q$. Then $P(x(k+1) \in H(s')|x(k))$ can be expressed as follows:

$$P(x_l^{(q)} + L^{(q)}(B(s')^T - \mathbf{1}) - x(k) \leq \tilde{\xi}_q \leq x_l^{(q)} + L^{(q)}B(s')^T - x(k)) = \tag{2.34}$$

$$F_{\tilde{\xi}_q}(x_l^{(q)} + L^{(q)}B(s')^T - x(k)) - F_{\tilde{\xi}_q}(x_l^{(q)} + L^{(q)}(B(s')^T - \mathbf{1}) - x(k)) = \tag{2.35}$$

$$F_{\tilde{\xi}_q,u}(B(s')^T, x(k)) - F_{\tilde{\xi}_q,l}(B(s')^T, x(k)) \tag{2.36}$$

Thus:
$$P(s, s') = \frac{1}{|H(s)|}\int_{H(s)} \left[F_{\tilde{\xi}_q,u}(B(s')^T, x(k)) - F_{\tilde{\xi}_q,l}(B(s')^T, x(k))\right] dx(k) \tag{2.37}$$

Consider now the case where $mode(s) \neq mode(s')$. Let $mode(s) = q$ and $mode(s') = q'$. Following a similar procedure, we can compute the transition probability as follows:

$$p_{qq'} = \frac{1}{|H(s)|}\int_{H(s)} S(q, q', x(k))dx(k) \tag{2.38}$$

$$P(s, s') = \frac{p_{qq'}}{|H(s)|}\int_{H(s)} \left[F_{\tilde{\eta}_{qq'},u}(B(s')^T, x(k)) - F_{\tilde{\eta}_{qq'},l}(B(s')^T, x(k))\right] dx(k) \tag{2.39}$$

The reachability algorithm is the same as in the case of general dynamical systems. However, there is no need to learn transition probabilities using sampling as closed form expressions are available. Notice, that the integrals appearing in the expressions of the transition probabilities might have to be computed numerically. In some special cases, closed form expressions can be derived for such integrals.

Chapter 3

Stochastic analysis of the thermal management system

We present the results obtained by the reachability algorithm described in Chapter 2 on the model presented in Chapter 1. We first introduce some application specific adjustment to the model. Then, we present a simulation result and the computation of some probabilistic metrics using both Monte Carlo methods and the method presented in this report. All the results have been obtained using the USHVER tool developed at the United Technologies Research Center Inc., under this contract.

3.1 Experiment setup

The system that we consider has five continuous variables and eight modes, although the last mode has a trivial behavior. The continuous variables are the altitude (h), the velocity (v), the fuel-tank mass (M), the fuel-tank temperature (T) and a clock variable (δ). We are interested in computing the marginal probability distribution of the fuel-tank temperature at the end of the mission (i.e. when the system enters the eighth mode). The dynamics of the height and velocity variables for the various modes are setup as described in Chapter 1. The clock variable evolves according to $\dot{\delta} = 1$. Some of the mode transitions are triggered by the clock variables. For example, the switching probability from the *taxing* mode to *take-off* mode depends only on the clock variable δ which is reset to zero after mode transition. The continuous dynamics for fuel-mass(M) and fuel-tank-temperature(T) for all modes are given as

$$\begin{aligned} \dot{M} &= -m_f \\ \dot{T} &= \frac{1}{M}\left(m_{in}T_{in} - m_{out}T + m_f T\right) \end{aligned} \tag{3.1}$$

where m_f is the rate at which fuel is consumed. Flow rate m_{out} is the rate at which fuel is drawn from the fuel-tank and m_{in} is the rate at which fuel is returned to the fuel-tank after re-circulation. The rate of fuel-consumption depends on the thrust according to the following equation:

$$m_f = \frac{0.7 f_{thrust}}{3600.0} kg/s. \tag{3.2}$$

The thrust requirement is fixed for each mode as described in Chapter 1. For the sake of this first set of results, we set $m_{out} = 2m_f$ and $m_{in} = m_{out} - m_f$. This means that we are not modeling any additional control on the fuel rate. A controller could be added to the model, albeit an increase in the number of states and, therefore, in the complexity of the analysis. The fuel absorbs heat from the fuel-oil heat exchanger before being consumed by the combustor. The temperature of the fuel

that is consumed by the combustor after passing through the fuel-oil heat exchanger(T_f) is given by the following equation:

$$T_f = \frac{H_L}{m_{out} C_{sp}} + T. \tag{3.3}$$

C_{sp} is the specific heat of fuel and is assumed to be 0.2 kJ/kg K. H_L is the heat-load generated by the EPS for each mode. Part of the fuel that is not consumed by the combustor is recirculated through the fuel-air heat exchanger back to the fuel-tank. The temperature drop in the fuel after passing through the fuel-air heat exchanger is assumed to be a fraction (f) of the difference in the temperature of the fuel and air. This temperature drop is a function of the heat-exchanger effectiveness and depends on how often the ram air inlet can be opened. We set $f = 0.1$. Thus the temperature of the fuel that goes back into the fuel-tank is given as

$$T_{in} = T_f + f(T_{air} - T_f) \tag{3.4}$$

The outside air temperature are modeled using different layers with different lapse rates. For our studies, only the first two layers are important. The temperature in the first two layers are given by the following model:

$$\begin{aligned} T_{air} &= T_0 - 6.5h \quad \text{for } 0 \leq h \leq 11.0 \\ T_{air} &= 0.751865 T_0 \quad \text{for } 11.0 < h \leq 20.0 \end{aligned} \tag{3.5}$$

where $T_0 = 15^0 C = 288.15K$ and h is in km. We assume an that there is an initial uncertainty in the fuel temperature. The fuel temperature is uniformly distributed between 288 and 298 degrees. Also, we discretize the dynamics of the system using a Euler Forward Scheme. The discretization step is 1 second.

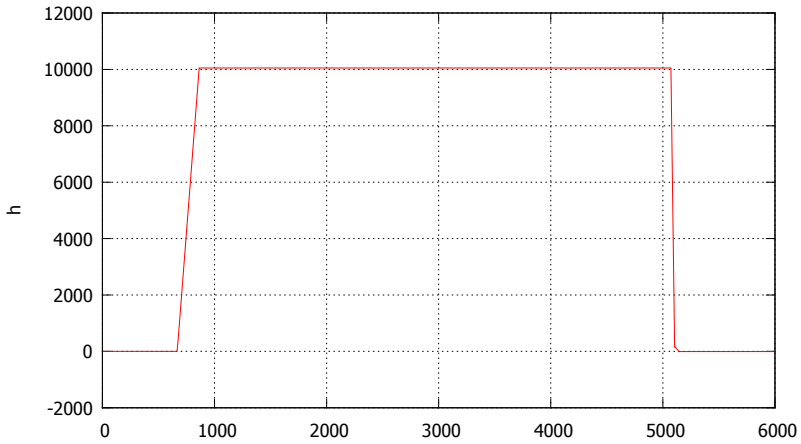
3.2 Simulation of the model

A simulation trace of the system obtained with USHVER is shown in Figure 3.1. The figure shows variables h, M and T as a function of time.

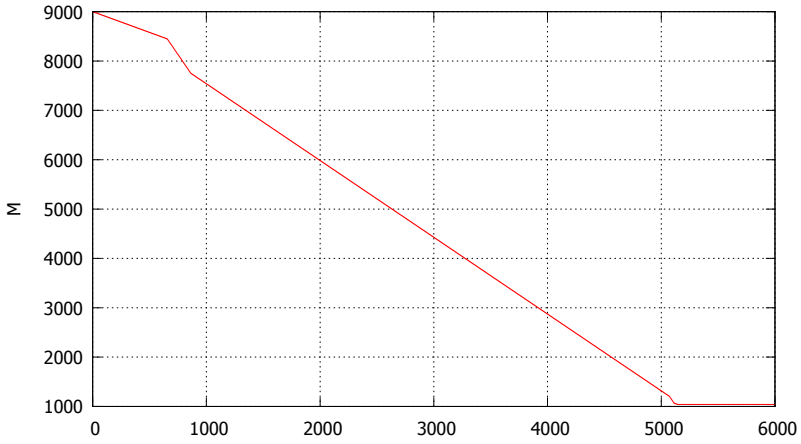
The take-off and ascending phase account for a small fraction of the mission. One may wonder if such modes could be abstracted by simply lumping the total fuel consumed during these two phases, and increasing the temperature of the remaining fuel. Although this two phases are executed at the highest fuel consumption rate (as can be seen from from mass M diagram), their precise dynamics is not interesting for the analysis and therefore they could indeed by abstracted into a discrete jump. As a consequence, the complexity of the analysis problem would be reduced as part of the state space does not need to be explored.

It is interesting to observe how the fuel temperature increases according to a non-linear law. During the execution of the mission, the fuel mass decreases. Thus, the fuel temperature increases more rapidly. However, for the first 3000 seconds the temperature profile appears to be fairly linear. Thus, it would be possible to approximate the fuel temperature dynamics by a clock variable $\dot{T} = c$, for $t \in [0, 3000]$, where c needs to be determined. This approximation would allow to treat all the variables as clocks and therefore providing a significant analysis speed-up. Moreover, when only clocks are present, analytical results are also available.

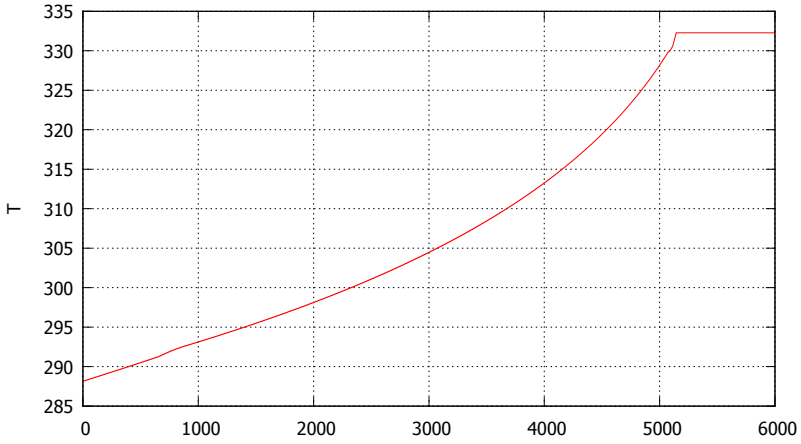
These types of simplifications are model and analysis dependent. They are usually suggested by either the system modeler or experts with domain knowledge. There are also techniques to abstract the dynamics of a hybrid systems by splitting modes into sub-modes where the dynamics can be approximated by clocks [28].



(a) Altitude profile.



(b) Fuel mass profile



(c) Fuel temperature profile

Figure 3.1: Results obtained using the USHVER simulator

3.3 Monte-Carlo Simulation

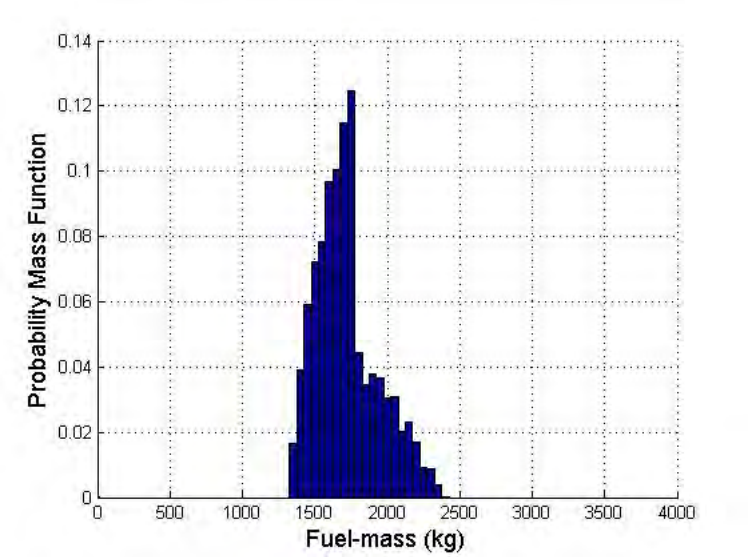
USHVER provides a Monte-Carlo simulator which can be instantiated on a DTSHS model to generates sample traces and gather statistics. The traces are generated by sampling the set of initial states according to the initial distribution $Init$. Other uncertain parameters are samples during the simulation. For example, each random trace corresponds to a unique pair of values for the taxing time and mission time which are chosen randomly. We generate 5000 such traces and for each random trajectory generated for the DTSHS, we store the values of all the state variables at the end of the mission. Figure 3.2 shows the marginal probability distributions for the mass(M) and temperature(T) of the fuel left in the fuel-tank at the end of the mission.

3.4 Reachability Analysis

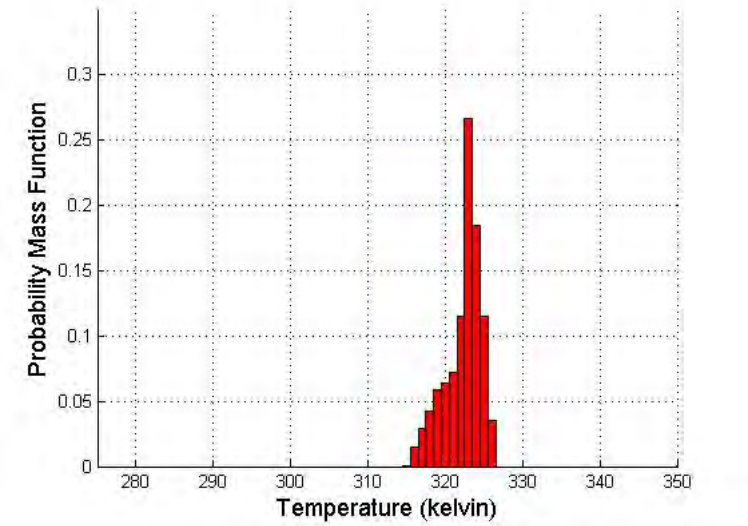
For reachability analysis, the continuous state space is partitioned into rectangular sets and each cell in the partition is encoded as described in Chapter 2. The number of grid points for each variable are chosen manually. The grid size should be chosen so that it captures the dynamics of the original system realistically. But also the grid size should be small enough so that the resulting data structures satisfy the memory constraints and so that the reachability computation can be done in a reasonable amount of time. Provided that some restrictions on the dynamics and probability distributions hold, the authors in [5] shows how the error in the probability computation can be bounded for a given partition of the continuous state space. Table 3.1 reports the list of parameters relative to the discretization grid. For each mode and for each variable, the table lists the minimum and maximum values of the variable in that mode, the number of selected grid points and the size of the intervals. Some of the invariants can be computed analytically whereas some others are learned through simulation. Selecting the grid size automatically is a challenge and requires further research.

Figure 3.3 shows the marginal probability distributions for the mass(M) and temperature(T) variables at the end of the mission. There are some differences in the marginal probability distributions obtained by Monte-Carlo simulations (Figure 3.2) and the reachability algorithm. We believe that this is partly due to undersampling in the Monte-Carlo simulations and partly due to the error introduced in the reachability algorithm by the discretization of the continuous state space.

Concluding Remarks. We have been able to successfully apply our probabilistic reachability algorithm to a DTSHS model of a thermal management system with 5 continuous variables and 7 modes. The execution of the reachability analysis and measure propagation takes roughly half an hour on a laptop with a Intel Core2 Duo CPU P9400 running at 2.40 GHz, and 2.95 GB RAM. Up to 3 million states were generated by the reachability computation. Areas for improvement include the automatic generation of the grid and the computation of rigorous confidence bounds for the results generated by the reachability algorithm for which we could leverage the work in [5].



(a) Marginal probability distribution for mass of fuel left in the fuel-tank at the end of the mission

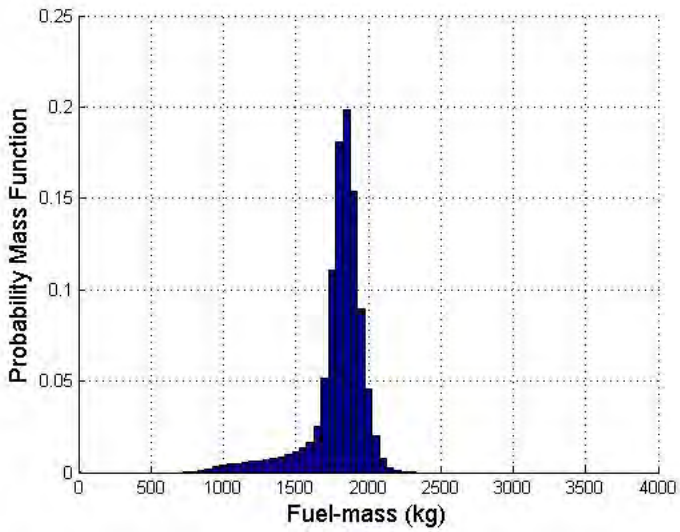


(b) Marginal probability distribution for the temperature of the fuel in the fuel-tank at the end of the mission

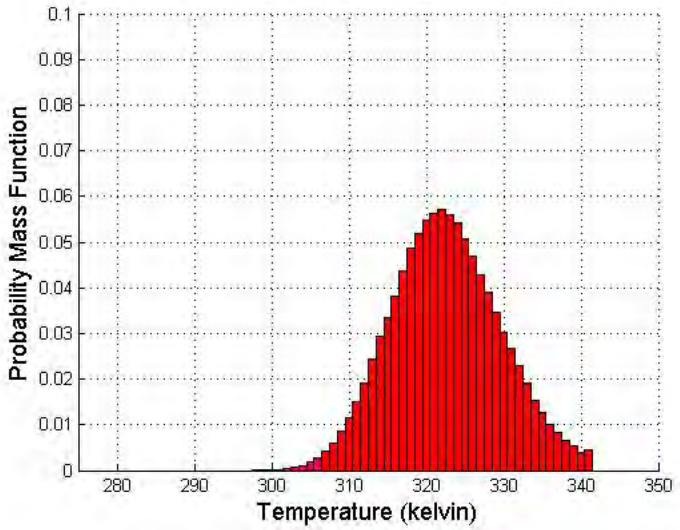
Figure 3.2: Results obtained using Monte Carlo simulations

Mode	Var.	$x_{d,l}^{(i)}$	$x_{d,u}^{(i)}$	$g_d^{(i)}$	$l_d^{(i)}$
0	h	0	10050	1	10050
	v	0	102.6	1	102.6
	M	7834	9000	110	10.6
	T	288	291.4	3	1.13333
	δ	0	650	900	0.722222
1		0	10050	1	10050
		0	75	10	7.5
		7820	8454	5	126.8
		291.05	291.45	2	0.2
		0	659	1	659
2		0	12000	10	1200
		0	102.6	10	10.26
		7512	8446	140	6.67143
		291.1	292.4	2	0.65
		0	857	1	857
3		0	10052	1	10052
		0	102.6	1	102.6
		946	8280	1400	5.23857
		292.1	333	25	1.636
		0	4482	4500	0.996
4		0	10052	10	1005.2
		62.55	102.6	1	40.05
		882	2176	22	58.8182
		317.65	333.4	2	7.875
		0	4375	1	4375
5		0	10052	1	10052
		10	102.6	10	9.26
		876	2176	22	59.0909
		317.65	333.9	5	3.25
		0	4383	1	4383
6		-50	10052	5	2020.4
		62.55	102.6	1	40.05
		864	2176	5	262.4
		317.65	334.85	5	3.44
		0	4414	5	882.8
7		0	10052	1	10052
		62.55	102.6	1	40.05
		484	2176	1	1692
		317.65	345	1	27.35
		0	4468	1	4468

Table 3.1: tbl:griddy



(a) Marginal probability distribution for mass of fuel left in the fuel-tank at the end of the mission



(b) Marginal probability distribution for the temperature of the fuel in the fuel-tank at the end of the mission

Figure 3.3: Results obtained using the Probabilistic Reachability algorithm.

Chapter 4

Adding details to the thermal management system

In Chapter 1 we developed a model of the system under study as a single DTSHS. We introduce DTSHSs with inputs and outputs and we define their composition. We then develop a new model of the original system as composition of two controllers and a dynamical model of the TMS. We use this new model to also determine the minim amount of heat that needs to be rejected using ram air. Finally, we determine the minimum requirements that the embedded platform need to satisfy in order to support the control functions. These requirements can then be used to design and embedded architecture able to support the control function. The embedded architecture will be verified using a different set of tools that are better suited for these type of systems.

4.1 Input-Output DTSHS

Systems featuring inputs and output present some compositionality challenges when the output value at instant k depends directly on the input value at instant k. To avoid this situation, our definition of IODTSHS is such that the output value at instant k only depends on the value of the state at instance k. This is not a limitation and the two definitions can be proved to be equivalent.

In the sequel, we will use the following notation. For a real variable x, a discrete evaluation function is a mapping $V_x : \mathbb{N} \to \mathbb{R}$, such that given a discrete time instant k, the value of x at k is $V_x(k)$. We use the shorthand notation x_k to denote $V_x(k)$. Similarly, for a set of variables X, a valuation function is a mapping $V_X : \mathbb{N} \to [X \to \mathbb{R}]$ from the set of natural numbers to the set of functions that map variables in X to their values. We also use the shorthand notation X_k to denote $V_X(k)$.

Definition 7. *An Input-Output Discrete Time Stochastic Hybrid System (IODTSHS) is a tuple* $H = (U, O, Q, d, T, L, R, out, init)$ *where :*

- $U = \{u_1, \ldots, u_m\}$ *is a set of input variables (or inputs for short)*
- $O = \{o_1, \ldots, o_n\}$ *is a set of output variables (or outputs for short)*
- $Q = \{q_1, \ldots, q_l\}$ *is a set of discrete modes*
- $d : Q \to \mathbb{N}$ *associates to each discrete mode the size of the continuous state space. This function implicitly defines the set of state variables* X_q *for each mode* $q \in Q$*. The hybrid state space* S *of the IODTSHS is defined as* $S = \cup_{q \in Q} \{q\} \times \mathbb{R}^{d(q)}$.

- $T = \{T_q\}$ *is a family of stochastic maps such that in mode* q, $X_{q,k+1} = T_q(X_{q,k}, U_k, \xi_{q,k})$, *where* $\tilde{\xi_q}$ *is a vector of i.i.d. processes.*
- $L = \{L_q : \mathbb{R}^{d(q)} \times \mathbb{R}^m \to Dist(Q)\}$ *is a family of switching functions that associate to each state* $X_{q,k}$ *in mode* q *a discrete probability distribution over the set of modes.* $L_q(X_{q,k}, U_k)(q')$ *is the probability of jumping from mode* q *to mode* q' *when the value of the continuous state is* $X_{q,k}$ *and the value of the input is* U_k.
- $R = \{R_q^{q'}\}$ *is a family of stochastic maps such that* $X'_{q',k} = R_q^{q'}(X_{q,k}, U_k, \eta_{q,k})$, *where* $\tilde{\eta_q}$ *is a vector of i.i.d. processes.*
- $out = \{out_q\}$ *is a family of stochastic maps such that* $U_k = out_q(X_{q,k}, \nu_{q,k})$, *where* $\tilde{\nu_q}$ *is a vector of i.i.d. processes.*
- $init : \mathbb{B}(S) \to [0, 1]$ *defines an initial probability distribution over the state space.*

The semantics of such model is similar to the one defined in Section 2.1.1 and we are not going to expand on it. The only difference that can be noted is the addition of the inputs. We also notice that the output functions do not take inputs as arguments.

We can now define a composition operation for IODTSHS. We introduce some notation that will turn out to be useful in the definition of the composition operator. Let X be a set of variables and $X' \subseteq X$. We define valuation projection as follows: $X_k|_{X'} = V_{X'}(k)$ such that for each $x \in X'$, $V_{X'}(k)(x) = V_X(k)(x)$.

Definition 8. *Let* $H^{(1)} = (U^{(1)}, O^{(1)}, Q^{(1)}, d^{(1)}, T^{(1)}, L^{(1)}, R^{(1)}, out^{(1)}, init^{(1)})$ *and* $H^{(1)} = (U^{(2)}, O^{(2)}, Q^{(2)}, d^{(2)}, T^{(2)}, L^{(2)}$ *be two IODTSHS. The composition* $H = H^{(1)}||H^{(2)}$ *is a IODTSHS defined only when* $O^{(1)} \cap O^{(2)} = \emptyset$ *and for all* $q \in Q^{(1)} \times Q^{(2)}$, $X_q^{(1)} \cap X_q^{(2)} = \emptyset$*:*

- $O = (O^{(1)} \cup O^{(2)}) \setminus (U^{(1)} \cup U^{(2)})$, $O^+ = O^{(1)} \cup O^{(2)}$
- $U = (U^{(1)} \cup U^{(2)}) \setminus O^+$, $U^+ = U^{(1)} \cup U^{(2)}$
- $Q = Q^{(1)} \times Q^{(2)}$ *is a set of discrete modes*
- $d : Q \to \mathbb{N}$ *such that* $d(q_1, q_2) = d^{(1)}(q_1) + d^{(2)}(q_2)$. *The state variables are considered to be* $X_q = X_q^{(1)} \cup X_q^{(2)}$
- $T_q(X_{q,k}, U_k, \Xi_{q,k})|_{X_q^{(i)}} = T_q^{(i)}(X_{q,k}|_{X_q^{(i)}}, U_k^+|_{U^{(i)}}, \Xi_{q,k}|_{\xi_q^{(i)}})$, $\forall q \in Q$, $\forall X_{q,k} \in \mathbb{R}^{d(q)}$, $\forall U_k \in \mathbb{R}^m$, $i = 1, 2$, *where* $\Xi_q = \xi_q^{(1)} \cup \xi_q^{(2)}$
- $L_q(X_{q,k}, U_k) = L_q^{(1)}(X_{q,k}|_{X_q^{(1)}}, U_k^+|_{U^{(1)}}) L_q^{(2)}(X_{q,k}|_{X_q^{(2)}}, U_k^+|_{U^{(2)}})$, $\forall q \in Q$, $\forall X_{q,k} \in \mathbb{R}^{d(q)}$
- $R_q^{q'}(X_{q,k}, U_k, \Theta_{q,k})|_{X_q^{(i)}} = R_q^{q',(i)}(X_{q,k}|_{X_q^{(i)}}, U_k^+|_{U^{(i)}}, \Theta_{q,k}|_{\eta_q^{(i)}})$, $\forall q,\ q' \in Q$, $\forall X_{q,k} \in \mathbb{R}^{d(q)}$, $\forall U_k \in \mathbb{R}^m$, $i = 1, 2$, *where* $\Theta_q = \eta_q^{(1)} \cup \eta_q^{(2)}$
- $out_q(X_{q,k}, \Gamma_{q,k})|_{U^{(i)}} = out_q^{(i)}(X_{q,k}|_{X_q^{(i)}}, \Gamma_{q,k}|_{\nu_q})$, $\forall q \in Q$, $\forall X_{q,k} \in \mathbb{R}^{d(q)}$, $i = 1, 2$, *where* $\Gamma_q = \nu_q^{(1)} \cup \nu_q^{(2)}$
- $init = init^{(1)} \times init^{(2)}$ *is the product distribution of the initial distributions of the two IODTSHSs being composed.*

The composition operator renders our model compositional with respect to the behavior of the system meaning that the behavior of the composed system can be derived from the behavior of the components. When the composition results in a closed system (i.e. a system without inputs) then the composed system is equivalent to the model presented in Section 2.1.1. We have implemented the composition operator in USHVER and we also provide the ability to define systems hierarchically. Hierarchy is a structural concept. In our framework, the hierarchy is defined by the parse tree of the composition expression. For example, consider the IODTSHS defined as $H = (H_1||H_2)||(H_3||H_4)$. Then, the system hierarchy is as follows: System H contains two sub-systems H' and H'', where H' contains sub-systems H_1 and H_2, and H'' contains sub-systems H_3 and H_4.

Using this new model, we will now refine the system presented in Chapter 1 into the composition of three sub-systems: a model of the dynamics of the the system, a controller of the fuel rate and a controller of the ram air inlet (open or close).

4.2 A new model for the TMS case study

The new model we use for our case study is shown in Figure 4.1. This new system comprises three sub-systems:

- *TMS* is the thermal management system. It has two outputs $O = \{T_f, T_out\}$ where T_f is the fuel temperature right before entering the combustor, and T_out is the fuel temperature in the tank. In our model, these two variables are also state variables. Therefore, the output function of this system is simply an identity function.
- *MDOT Controller* is a controller that regulates the rate of the fuel that exists the fuel tank. The fuel rate is regulated to maintain temperature T_f within some given bounds.
- *RA Controller* is a controller that decides whether the ram air inlet is open or close. This controller tries to maintain the fuel temperature within acceptable limits.

The two controllers are implemented here independently. However, we notice that this is not the best solution for this system. In fact, in order to lower the fuel temperature in the tank, the fuel rate must be higher than $\dot{m}_f$ so that some fuel is allowed to circulate through the fuel/air heat exchanger. The joint optimization of the two controllers is the subject of Chapter 7, while in this chapter we limit ourselves to two simple control strategies to illustrate some of the capabilities of our design system.

The ram air controller has only two modes $Q^{(RA)}\{q_o, q_c\}$. In mode q_o, the output has value 1, meaning that the ram air inlet is open. In mode q_c, the output has value 0, meaning that the ram air inlet is closed. There is no dynamics associated with the two states. The switching functions are defined as follows:

- $L^{(RA)}_{q_o}(T_{out,k})(q_c) = 0$ if $T_{out,k} \geq T_{out,min}$, and $L^{(RA)}_{q_o}(T_{out,k})(q_c) = 1$ if $T_{out,k} < T_{out,min}$
- $L^{(RA)}_{q_c}(T_{out,k})(q_o) = 0$ if $T_{out,k} \leq T_{out,max}$, and $L^{(RA)}_{q_c}(T_{out,k})(q_o) = 1$ if $T_{out,k} > T_{out,max}$

This is an implementation of a bang-bang controller with hysteresis.

The flow rate controller is a simple proportional controller with bounds. The controller has only one mode and the dynamics is defined as follows:

$$\dot{m}_{out,k+1} = \min\{\dot{m}_{max}, \max\{0, \dot{m}_{out,k} + \alpha \cdot (T_{f,k} - T_d)\}\}$$

where α determines the bandwidth of the controller and T_d is the desired temperature. We note that this is the flow rate in excess of $\dot{m}_f$. This model is highly non-linear and the two bounds constraint the ability of the system to regulate the temperature at the combustor. Thus, a critical design decision is the amount of heat to be rejected by the fuel/air heat exchanger.

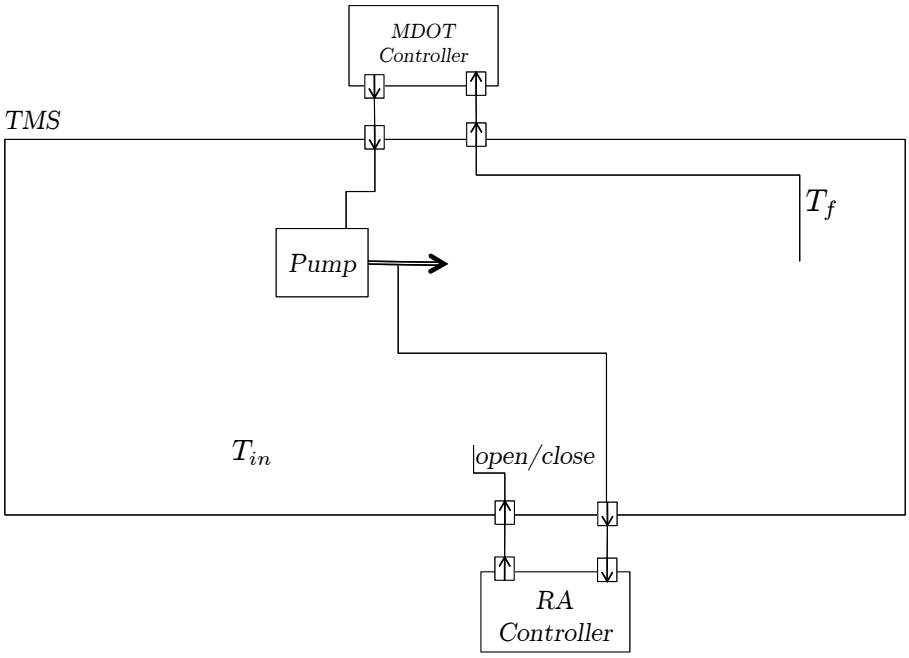


Figure 4.1: Refinement of the first level model and decomposition into three sub-systems including two controllers for the flow rate and ram air inlet.

4.3 Impact of the heat exchanger efficiency

The purpose of this section is to determine the amount of heat that needs to be rejected by the thermal management system through ram air. We could in principle perform a probabilistic analysis and determine the minimum probability that that the output of the ram air controller be equal to 1. This would then correspond to changing the switching functions $L^{(RA)}$ in order to guarantee such probability. However, this type of analysis makes little sense in this context. A better way of defining the amount of heat to be rejected by ram air is to compute the effectiveness of the fuel/air heat exchanged. For this analysis, we simply use simulation. This is going to be a worst case analysis for maximum taxi time (600 s) and maximum flying time (4800 s). For this simulation, we fix $T_{out,min} = 280°K$, $T_{out,max} = 290°K$, $T_d = 340°K$, $\alpha = 1$. There is no guarantee that the temperature in the fuel tank will be in the range $[T_{out,min}, T_{out,max}]$ since this depends on the balance between the generated heat and the ability to reject it. For the same reason, there is no guarantee that the fuel temperature at the combustor will be close to T_d.

Figure 4.2 shows the impact of the effectiveness of the heat exchanger on the temperatures in the fuel tank T_{out} and at the combustor T_f. The value of the effectiveness is swept from 0.2 to 0.45 and we report the maximum and minimum values for the temperatures. The point in time at which the maximum and minimum values are attained change depending on the effectiveness. The two plots at the bottom show how temperature changes over time. Notice that the temperature values until take off is not affected by the effectiveness because ram air cannot be used (we assume that there are no fans to force an airstream in the inlet). When the effectiveness is 0.2, the maximum value is attained at the end of the mission, whereas when the effectiveness is 0.45, the maximum value is attained right before taking off. Based on this analysis, we select an effectiveness value of 0.35.

Remark 1. *The way in which this effectiveness is achieved is not specified here. In general, there could be different ways of achieving it. In some applications, the use of bleed air is not advisable for*

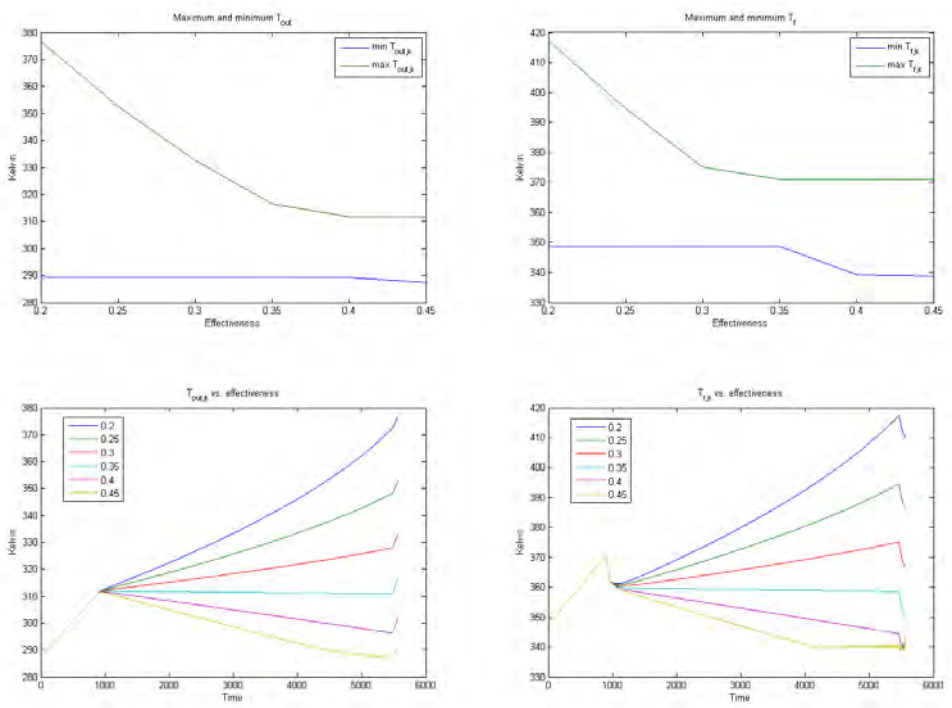


Figure 4.2: Impact of the effectiveness of the fuel/air heat exchanger on the maximum and minimum temperatures of the fuel in the tank and at the combustor.

efficiency reasons. Drawing bleed air from the engines represent a loss that inevitably compromises the efficiency metric. In some other cases, bleed air can be used as a way of colling down some of the heat loads. In this case, there could be a decision to be made whether one or the other should be used. Although this control design effort was originally included in the statement of work, we feel that a much more interesting (and complex) control de sign problem is to decide on the optimal fuel flow rate over the entire mission so that the temperature at the combustor is close to a given ideal value (Chapter 7).

4.4 Taking into account architectural metrics

The analysis of the architecture supporting the control functions is a rather different problem that the one presented so far. Analysis of the performance of an architecture that comprises computation and communication components can be either deterministic or stochastic. In Chapter 8, we review the work we have done in the probabilistic setting while we refer the reader to some other general techniques that are available for other types of analysis. The probabilistic setting also allows to model faults that may affect the performance of the system.

There are two different problems that can be addresses. The analysis problem has the following typical setup. A performance target is given for an architecture, such as the minimum throughput of a connection or the maximum delay between two events. The architecture is then analyzed and the result of the analysis is checked against the given target. The synthesis problem, instead, starts

from the performance target and uses a procedure (or algorithm) to come up with an architecture that matched the given performance requirements. In both cases, the performance goal has to be defined. It can be defined based on an intuition of what the application that will be running on the platform needs, or it can be derived in a more formal way.

Ideally, the control application should be analyzed with some of the platform induced delay injected in the model. Those delays are the assumption on the platform and, thus, its specification. This way, the control algorithms will be designed to be robust to those delays. The designers will have to balance the delay values (that that would like to be zero) with the complexity of designing the underlying embedded platform.

There are several ways of injecting platform induced delays in the functional model. One approach as been also recently proposed by []. In our framework, we can model deterministic as well as probabilistic delays which allows to analyze also the impact of jitter on control algorithms. At the functional level, delays can be typically captured using clocks. For probabilistic delays and for expo y not need cient to ca

a) *b)*

Figure 4.3: Example of refinement of the functional model to account for platform induced delays.

The model of a controller can then be enriched or refined as shown in Figure 4.3. The original model needs to be refined by taking into account three sources of delays: the delays associated with receiving data from sensors, sending data to actuators, and computing the control function. Communication delays (shown in Figure 4.3.b)represent refinements of the ideal communication channels (drawn as simple lines in our diagrams) with a more complex channel that induces a delay on data. The computation of the control function should also be annotated with delays because it will ultimately take time to check for guard conditions and to compute next states and outputs. In this simple example, the control function reduces only to checking that guards conditions are satisfied. The refinement we show in figure refers to geometrically distributed transition times.

The analysis of these new type of models may become prohibitively complex in the general case. Communication delays are typically modeled using queues. Because each element of the queue is a continuous state, the number of states become larges as the delay increases. Computation delays

can be modeled using clock variables whose values are used to guide transitions. Although clock variables are easier to handle than general state variables, they also add to the complexity of the analysis problem. We foresee simulation based methods to be more effective in analyzing these models.

Chapter 5

Coping with complexity through an organized design flow

The complexity of the analysis task for DTSHS requires to develop strategies that are able to decompose the analysis into sub-tasks of manageable complexity. This can be achieved with the support of a design flow that proceeds by refinement. In this chapter we explore the use of contract based design [31, 14] for probabilistic systems and we apply the methodology to the refinement of the heat load sub-system and the heat sink in the next chapter. We start with an introduction to contract-based design and we explain what are the challenges in applying this type of reasoning to probabilistic systems. We then show some application scenarios and we go into an in depth definition of the probabilistic contracts that we will adopt. For these contracts, we define the two basic operation of composition and refinement that allow compositional reasoning.

5.1 Introduction to Contract-Based Design

We will use the notion of traces to define models and contracts [22, 24]. Let V be a set of variables. A valuation is a function $V \to D$ where D is the domain of the variables. Then a behavior is a sequence of valuations, i.e. $\{\sigma_i\}$, $\sigma_i \in [V \to D]$. The set of all possible finite and infinite behaviors over the set of variables V is simply denoted $\Sigma(V) = [V \to D]^\infty$. This is a denotation definition of behaviors. A set of behaviors can be specified in practice using operational models, or expressions. In this cases, the set of behaviors is defined by the set of legal execution of the model (e.g. the language of a state machine), of by the set of traces that satisfy the expression, respectively.

A component $M = (V, B)$ is characterized by a set of variables[1] V and a set of behaviors $B \subseteq \Sigma(V)$. We simplify the notation by using M to denote the set of behaviors of the model, and V_M for its variables.

A contract is a triple $C(V, A, G)$ where V is a set of variables, $A \subseteq \Sigma(V)$ is called assumption, and $G \subseteq \Sigma(V)$ is called guarantee. The assumption is a set of behaviors that define the contexts in which a component is operates. The guarantee is the set of behaviors that the component promises if the assumptions are satisfied. The assumption and the guarantee are defined on the same set of variables.

Given a model M and a contract $C(V_M, A, G)$, then the model satisfies the contract if and only if:

$$M \cap A \subseteq G \tag{5.1}$$

Given a contract $C(V, A, G)$, there exist a unique maximal implementation $M_C = G \cup \neg A$ (where

[1] We will distinguish input and output ports later in this section

$\neg A$ is the complement of A, i.e. $\Sigma(V) \setminus A$). A model M satisfies a contract C, denoted $M \models C$, if and only if $M \subseteq M_C$.

Before defining composition and containment of contracts, we first define input and output ports of a model (and of a contract). We will equivalently refer to controlled and uncontrolled variables[15], respectively. Let the variables of models be partitioned in the set U of uncontrolled variables (i.e. those variables that are under the control of the environment) and X of controlled variables (i.e. those variables that are under the control of the model). The controlled variables can be further partitioned into the set of internal variables (not visible from the outside), and output variables. This distinction is, however, not essential for our discussion.

Example 1: Models and contracts
In this example, we refer to a clocked system where variables are real-values, and an execution is driven by a clock. Thus, $\Sigma(V) = [V \to \mathbb{R}^{|V|}]^\infty$. Sets of behaviors are simply defined by inequalities and equalities variables that are to be considered valid at each clock tick. Consider an adder components M with $U_M = \{i_1, i_2\}$ and $X_M = \{o\}$. The set of behaviors of the added is defined by $M = \{o = i_1 + i_2\}$. Consider now a contract for this component specified as follows:

$$A = \{0 \leq i_1 \leq 0.5, 0 \leq i_2 \leq 0.5\}, \quad G = \{o \leq 10\}$$

First we need to compute the intersection of M with the assumption A. This intersection is the following system of relations:

$$\begin{aligned} o &= i_1 + i_2 \\ 0 &\leq i_1 \leq 0.5 \\ 0 &\leq i_2 \leq 0.5 \end{aligned}$$

which means $M \cap A = \{0 \leq o \leq 1\ \}$. All the traces that satisfy this expression, certainly satisfy G which implies $M \cap A \subseteq G$. □

One may wonder if this type of formalism is limited to discrete systems. The work in [13] shows that the same formalism applies to continuous time and hybrid systems by carefully defining the domain of the variables.

Consider two contracts $C_1(V_1, A_1, G_1)$ and $C_2(V_2, A_2, G_2)$. The composition of these two contracts is defined only when $X_1 \cap X_2 = \emptyset$, meaning that any two systems implementing the contracts do not control the same variables.

Definition 9 (Contract composition)**.** *Given two contracts $C_1(V_1, A_1, G_1)$ and $C_2(V_2, A_2, G_2)$ such that $X_1 \cap X_2 = \emptyset$, the parallel composition of the contracts is $C = C_1||C_2$ such that :*

- $X = X_1 \cup X_2$ *: the set uncontrolled variables is the union of the controlled variables of the two contracts.*
- $U = (U_1 \cup U_2) \setminus X$ *: the set of uncontrolled variables is the union of the set of uncontrolled variables of the two contracts, except those variables that are connected in "feedback".*
- $V = X \cup V$
- $G = G_1 \uparrow^V \cap G_2 \uparrow^V$ *: the composition must satisfy both guarantees.*
- $A = A_1 \uparrow^V \cap A_2 \uparrow^V \cup \neg G$ *: the composition must satisfy both assumptions. However, the assumption are also partially satisfied by the interaction among the two components. Thus, the assumption of the composite can be relaxed by adding $\neg G$.*

In this definition we used the notion of inverse projection of variables. For a set of behaviors B on variables V, the inverse projection over $V' \supseteq V$, denoted by $B \uparrow V'$ is the set of behavior $\{b \in [V' \rightharpoonup D']^\infty | b \downarrow V \in B\}$. The inverse projection is defined in terms of the projection operator $\downarrow$. The projection operator for contracts is contravariant for assumptions and guarantees [13]. Given a contract $C(V, A, G)$ and a variable $v \in V$, projection is defined as follows:

$$C \downarrow_v = (\forall v.A, \exists v.G)$$

Another type of composition is the conjunction of contracts. Conjunction is an operation that joins together two aspects of a model that are orthogonal. For example, one contract might be used to define the assumption and guarantees on the sequences of values, while another contract might be used to define the timing properties (in terms of real-time) of the sequences of values.

Definition 10 (Conjunction of contracts)**.** *Given two contracts $C_1(V_1, A_1, G_1)$ and $C_2(V_2, A_2, G_2)$ such that $V_1 = V_2$, their conjunction is a contract $C = C_1 \wedge C_2$ such that :*

- $V = V_1 = V_2$
- $A = A_1 \cup A_2$
- $G = G_1 \cap G_2$

This definition can be actually generalized to the case where the sets of ports are different [13]. The general definition used a pre-order on contracts. The pre-order capture the notion of refinement or substitutability. In words, a contract C_1 refines a contract C_2 is it makes more assumptions and provides less guarantees.

Definition 11. *Given two contracts $C_1(V_1, A_1, G_1)$ and $C_2(V_2, A_2, G_2)$ such that $V_1 = V_2$, contract C_1 refines contract C_2, denoted $C_1 \preceq C_2$ if and only if $A_1 \supseteq A_2$ and $G_1 \subseteq G_2$.*

5.2 Coping with complexity: compositional reasoning and design flows

The general definition of contracts leads to compositional rules for independent implementability. In particular, one key rule is the following:

Proposition 1. *Let M_1 and M_2 be two models (implementations) and, C_1 and C_2 be two contracts such that $M_1 \models C_1$ and $M_2 \models C_2$, then $M_1 || M_2 \models C_1 || C_2$.*

This rule finds numerous applications. During the design of a system, a system engineer could work with abstract models that satisfy some contracts. The system design activity goal is to satisfy a system goal expressed by a contract $C(A, G)$. Once the system has been analyzed such that C is satisfied, then the contracts for each sub-system can be delivered to the sub-system designers. If the refinement of each sub-system satisfies the prescribed contract, then the composition of the sub-systems should satisfy top level contract C.

Other rules are important to establish design flows based on contracts:

Proposition 2. *Let M be a model, and C_1 and C_2 two contracts such that $C_1 \preceq C_2$. Then $M \models C_1 \Rightarrow M \models C_2$.*

This proposition states that if a model satisfies a contract, then it satisfies any other less restrictive contract.

Remarks on contract-based design The definition of models and contracts has been given using a general denotational framework based on the notion of traces. In practice, an operational model for contracts needs to be used for automatic verification. Some models have been proposed that rely either on temporal logic, or directly on executable models. For example, the work in [15] uses hybrid systems to define contracts. In this chapter we have seen other formalisms to define contracts for probabilistic systems.

The way in which contracts are specified impacts the complexity of the parallel composition operator as well as the complexity of checking if a contract is a refinement of a more abstract contract. Unfortunately, the complexity refinement checking for the probabilistic contracts that we have reviewed in this section is high. The benefit of decomposing the analysis task may vanish unless a more practical approach can be found.

5.3 Contract-Based Design for probabilistic systems

The idea of contracts can be extended to probabilistic systems. Extending contracts to probabilistic systems is a non-trivial task. As discussed in Section 5.2, a contract specifies a set of behaviors as assumptions, and a set of behaviours as guarantees. In the context of probabilistic systems, a behaviour (or better a set of behaviors) has an associated probability measure (see for example [11] for the definition of the probability measure defined by a Markov chain). When implementing a sub-system that is required to satisfy a contract, one can choose to implement a tighter contract, i.e. one that makes more assumptions and less guarantees (although it might not be the most cost effective choice).

One might be tempted to define the assumptions and the guarantees as stochastic processes with a precise statistics. For example, consider contracts captured by Markov Decision Processes with precise probability distribution functions. The implementation of such contract must make sure to match such probability distribution precisely, which would be, indeed, a difficult task. In fact, the probability measure associated with the states of the system (and evolving in time) can be though of as an additional state variable of the system which needs to be "implemented" by the designer. For this reason, the approach that has been historically followed is to describe the specification of a probabilistic system (i.e. its contract) using probability intervals [3]. This approach as been used also in more recent works [53].

5.3.1 Probabilistic systems and probabilistic specifications

Jonssnon and Larsen [3] in 1991 already posed the problem of specification and refinement for probabilistic systems. In this work, a probabilistic specification involves the definition of intervals of probabilities. In summary, the transition between two states of a system has an associated interval of probabilities rather then a single probability value. This type of specification allows some flexibility when the system is refined into an implementation.

In this section we present a simplified description of the procedure as an introduction of other related works. Consider a probabilistic model defined by a tuple $M(S, P, V)$ where S is a set of states, $P : S \times S \to [0, 1]$ is a transition probability function, such that for all $s \in S$, $\sum_{s' \in S} P(s, s') = 1$, and $V : S \to 2^A$ is an output function with A a set of atomic propositions. With abuse of notation, we will use P also to denote the transition matrix of the probabilistic model such that $P_{i,j} = P(i, j)$. Also, for a set $Q \subset S$ we adopt the notation $P(s, Q) = \sum_{s' \in Q} P(s, s')$, $P(Q, s) = \sum_{s' \in Q} P(s', s)$,$P(Q, Q') = \sum_{s \in Q} \sum_{s' \in Q'} P(s, s')$.

Definition 12 (Bisimulation relation)**.** *Let $M(S, P, V)$ be a probabilistic model. An equivalence relation R on S is a probabilistic bisimulation on S if given two states $i, j \in S$ such that $(i, j) \in R$, the followings hold:*

- $V(i) = V(j)$,

n Unlimited.

0.8

(a) A probabil

(b) The abstraction of the model

- $\forall\ Q \in S/R,\ P(i,Q) = P(j,Q)$

Two states $i, j \in S$ are probabilistic bisimulation equivalent, written $i \simeq j$ if there exists some probabilistic bisimulation R such that $(i,j) \in R$.

The notion of bisimulation allows to group states that are indistinguishable to an observer. We illustrate this concept through an example.

Example 2: Bisimulation
Consider the probabilistic model in Figure 5.1(a). Each state is labelled with the state name and the set of atomic propositions that are true in that state. Let the bisimulation relation be $R = \{(s_1, s_3), (s_2, s_4)\}$. It is easy to verify that the conditions in Definition 12 are all satisfied:

- $V(s_1) = V(s_3)$ and $V(s_2) = V(S_4)$
- $P(s_1, \{s_2, s_4\}) = 0.7 = P(s_3, \{s_2, s_4\})$, and $P(s_2, \{s_0\}) = 0.8 = P(s_4, \{s_0\})$

When a bisimulaiton relation has been found, then one can analyze an abstraction of the system where only equivalence classes are considered (because states withing a class cannot be distinguished by an observed only looking at the sequence of atomic propositions). Figure 5.1(b) shows the abstracted system. There are only three states corresponding to the three equivalence classes induced by the bisimulaiton relation. There are several technical definitions of the equivalence of two models induced by a bisimulation relation. Our purpose is only to provide an intuition through an example. As a further exercise, we can for example compute the probability that starting from s_0, *end* will eventually become true (i.e. $Prob(true \mathcal{U} end)$). For the model in Figure 5.1(a), there are three paths that satisfy this condition: $s_0 \to s_1 \to s_2$ with probability 0.21, $s_0 \to s_3 \to s_2$ with probability 0.04, and $s_0 \to s_3 \to s_4$ with probability 0.0.03. Thus, the total probability is 0.28. For the model in Figure 5.1(b) there is only one path with probability 0.28 as well. □

A probabilistic specification is a tuple $\mathcal{S}(S, \mathbf{P}, V)$ where S is a set of states, $\mathbf{P} : S \times S \to 2^{[0,1]}$ is a probabilistic transition function that associates an interval of probabilities to each transition, and $V : S \leftarrow 2^A$ is an output function with A a set of atomic propositions.

It is a natural question to ask whether a probabilistic system satisfies a probabilistic specification. The satisfaction relation can be defined as follows

Definition 13 (Satisfaction relation)**.** *Let $\mathcal{S}(S, \mathbf{P}, V)$ be a probabilistic specification and $M(S_M, P_M, V_M)$ be a probabilistic model. A relation $R \subseteq S_M \times S$ is a satisfaction relation if $(s_m, s) \in R$ implies the followings:*

- $V_M(s_m) = V(s)$

ted.

0.1

(d) A model that satisfies the specification

- *There exists a probability distribution* $\delta : P \times S \to [0, 1]$ *such that:*
  - *for all* $s'_m \in S_M$, $\delta(s'_m, S) = P_M(s_m, s'_m)$
  - *for all* $s' \in S$, $\delta(S_M, s') \in \mathbf{P}(s, s')$
  - $\delta(s'_m, s') > 0 \Rightarrow (s'_m, s') \in R$

We will write s_m **sat** s *if and only if* $(s_m, s) \in R$ *for some satisfaction relation* R*.*

The probability distribution δ defines a relation between the sets of next states in the probabilistic specification and in the probabilistic model. This relations defines which part of a transition in the specification is assigned to what part of which transition in the probabilistic model. We clarify this concept through an example.

Example 3: Satisfaction
Figure 5.1(c) shows a probabilistic specification and Figure 5.1(d) a probabilistic model. We will show in this example the the probabilistic model satisfies the probabilistic specification. First, we provide a brief explanation of what the models represent. Consider a system characterized by three levels (or a representative variable): l is a em low level, m is a *medium* level, and h is a *high* level. The system is captured by three states s'_l, s'_m and s'_h, respectively. The transitions among these states are all intervals $[0.2, 0.8]$. One could interpret these probabilities as the speed at which the system transitions from one state to another. Also notice that we don't explicitly represent self transitions, but they are actually present in the model.

The model in Figure 5.1(d) captures the same type of system where fast and slow transitions between states are separated: s_{vl}, s_{vm} and s_{vh} represent states such that transitions are fast, while s_l, s_m and s_h represent states where transitions are slow. There is also a small probability that a system moving fast from a high level to a low level starts transitioning slow and viceversa.

We now check that the relation:

$$R = \{(s_{vl}, s'_l), (s_l, s'_l), (s_{vm}, s'_m), (s_m, s'_m), (s_{vh}, s'_h), (s_h, s'_h)\}$$

is a satisfaction relation. Consider the element $(s_{vl}, s'_l) \in R$. Figure 5.1 shows the way in which the probability distribution δ is represented: a dashed line between two states s and t indicates that $\delta(s, t) > 0$. The first condition is satisfied because $V(s_{vl}) = \{l\} = V(s'_l)$. To check the second condition, we only need to check the following:

$$P_M(s_{vl}, s_{vl}) + P_M(s_{vl}, s_l) = 0.2 \in \mathbf{P}(s'_l, s'_l) = [0.2, 0.8]$$

0.8

Figure 5.1: Example of weighting function.

Also, we have that $(s_l, s'_l) \in R$, $(s_{vm}, s'_m) \in R$ which satisfies the last condition of the satisfaction relation definition. One can check all the other pairs in R in a similar way:

$$
\begin{aligned}
&P_M(s_{vm}, s_{vm}) = 0.1 \in \mathbf{P}(s'_m, s'_m) = [0, 0.6] \\
&P_M(s_{vm}, s_{vl}) + P_M(s_{vm}, s_l) = 0.5 \in \mathbf{P}(s'_m, s'_l) = [0.2, 0.8] \\
&P_M(s_{vh}, s_h) + P_M(s_{vh}, s_{vh}) = 0.2 \in \mathbf{P}(s'_h, s'_h) = [0.2, 0.8] \\
&P_M(s_{vh}, s_{vm}) + P_M(s_{vh}, s_m) = 0.8 \in \mathbf{P}(s'_h, s'_m) = [0.2, 0.8] \\
&P_M(s_l, s_l) + P_M(s_l, s_{vl}) = 0.8 \in \mathbf{P}(s'_l, s'_l) = [0.2, 0.8] \\
&P_M(s_l, s_m) = 0.2 \in \mathbf{P}(s'_l, s'_m) = [0.2, 0.8] \\
&P_M(s_m, s_m) = 0.4 \in \mathbf{P}(s'_m, s'_m) = [0, 0.6] \\
&P_M(s_m, s_l) + P_M(s_m, s_{vl}) = 0.2 \in \mathbf{P}(s'_m, s'_l) = [0.2, 0.8] \\
&P_M(s_h, s_h) + P_M(s_h, s_{vh}) = 0.8 \in \mathbf{P}(s'_h, s'_h) = [0.2, 0.8] \\
&P_M(s_h, s_{vm}) + P_M(s_h, s_m) = 0.2 \in \mathbf{P}(s'_h, s'_m) = [0.2, 0.8]
\end{aligned}
$$

Thus R is a satisfaction relation $\square$.

The satisfaction relation can be used to define refinement among two specifications[2].

Definition 14 (Refinement)**.** *Let $\mathcal{S}(S, \mathbf{P}, V)$ be a probabilistic specification. Then $s \in S$ refines $t \in S$, written $s \subseteq t$ if and only if for any probabilistic model $M(S_M, P_M, V_M)$ and for any state $p \in S_M$, p **sat** $s \Rightarrow p$ **sat** t.*

This definition refers to refinement between two states in a specification as a relation where a state s is more refined than t if its specification is more "stringent" than t. This is because if a state p in a model is able to satisfy the specification state s, then it will be able to satisfy t as well. However, there might be models for which t can be satisfied by a state p which is not able to satisfy s.

This definition is somehow not operational, meaning that it will not allow to define a procedure to check refinement. The following definition (together with the subsequent proposition) give a way of checking refinement.

[2]Notice that a specification where the interval probabilities reduce to a single number can be casted to a probabilistic model. Thus, refinement is a notion that also extends to the case where a model refines a specification.

Definition 15 (Simulation relation)**.** *Let $\mathcal{S}(S, \mathbf{P}, V)$ be a probabilistic specification. A simulation R on $\mathcal{S}$ is a relation $R \subseteq S \times S$ such that $(s,t) \in R$ implies the followings:*

- $V(s) = V(t)$
- *There is a function $\rho : S \times S \to [0,1]$ such that:*
  - *for any function $f : S \to [0,1]$ such that $f(s') \in \mathbf{P}(s,s')$, and for any $t' \in S$:*

$$\sum_{s' \in S} f(s') \cdot \rho(s',t') \in \mathbf{P}(t,t')$$

  - $\rho(s',t') > 0 \Rightarrow (s',t') \in R$

t simulates s if there exists a simulation relation R that that $(s,t) \in R$.

Proposition 3. *Let $\mathcal{S}(S, \mathbf{P}, V)$ be a probabilistic specification and $s, t \in S$ two states, then t simulates $s \Rightarrow s \subseteq t$.*

This proposition gives us a way of checking refinement between probabilistic specifications (and therefore enables us to check whether a probabilistic model refines a probabilistic specification). One can imagine how this concepts could be used in a design flow. At the higher abstraction levels, a system can be analyzed using abstract specifications of components. Once the system has been proved correct, each component can be implemented separately as long as it satisfies the specification. This methodology can be effectively used only if checking properties on probabilistic specifications is not a complex task, and if also checking refinement is not prohibitively complex.

The two problems to be addressed then are the following:

- Given a probabilistic specification $\mathcal{S}(S, \mathbf{P}, V)$ and a property $\mathcal{P}$ expressed in some probabilistic logic, check whether $\mathcal{S}$ satisfied $\mathcal{P}$.
- Given two specifications $\mathcal{S}_1$ (abstract) and $\mathcal{S}_2$ (refined), find a simulation relation between the states of $\mathcal{S}_2$ and the states of $\mathcal{S}_1$ (i.e. check whether $\mathcal{S}_1$ simulates $\mathcal{S}_2$) which implies that $\mathcal{S}_2$ refines $\mathcal{S}_1$.

The first problem turns out to be complex. Consider a probabilistic specification $\mathcal{S}(S, \mathbf{P}, V)$ and a state s. The probability distribution associated with the transitions emanating from s is one of the distributions from the following set:

$$D_s = \{\mu \in Dist(S) | \mu(s') \in \mathbf{P}(s,s')\}$$

When checking if a specification satisfies a properties, all possible choices should be checked (because they are all possible refinements of the specification). For the type of specifications we are considering here, namely specifications where transition probabilities are intervals, there are algorithms that can solve this problem [3, 37]. However, the general problem is complex. Consider the reachability problem (which is a basic algorithm for checking properties).[3].

The second problem presents some difficulties but it can be solved using linear programming or network flow formulations [9, 10].

Remark 2 (Abstraction of a system into a specification)**.** *We have defined refinement for probabilistic specifications. The notion of refinement can also be used to define abstractions. In [37], it is shown how to generate an abstraction of a CTMC into an interval specification.*

[3] Alessandro Note: Need to describe an example that shows the complexity

5.3.2 Review of previous work on probabilistic contract-based design

Although the concepts of refinement and simulation for finite probabilistic systems has been around form many years, there are few approaches to define probabilistic contracts and to build a framework for probabilistic contract-based design (PCBD). In this section we review the most recent work and we also review related approaches.

PCBD using Interactive Markov Chains Xu *et al* [53] propose a framework for probabilistic contract based design which is based the Interactive Markov Chains (IMC) model. This type of models include both deterministic transitions driven by events, and probabilistic transitions.

Definition 16. *An IMC is a tuple* $(S, A, \rightarrow, \pi, s_0)$ *where:*

- S *is a non-empty set of states partitioned into* S^p*, the set of probabilistic states, and* S^a *the set of action states;*
- A *is a finite set of actions;*
- $\rightarrow \subseteq S^a \times A \times S$ *is an action transition relation ;*
- $\pi : S^p \rightarrow (S \rightarrow [0,1])$ *is a transition probability function such that for each* $s \in S^p$*,* $\pi(s)$ *is a probability distribution over* S*;*
- $s_0 \in S$ *is the initial state.*

This model is an extension of the probabilistic model defined in Section 5.3.1. The extension is in the addition of the action set and the action transition function. Notice that the set of states is partitioned such that actions are only "visible" in action states. However, chains of probabilistic states or actions states are possible in the model.

Definition 17. *An contract is a tuple* $(S, A, \rightarrow, \sigma, s_0)$ *where:*

- $S = S^p \cup S^a \cup \{\bot\}$ *is a non-empty set of states partitioned into* S^p*, the set of probabilistic states, and* S^a *the set of action states, and an accepting state* $\bot$ *without outgoing transitions;*
- A *is a finite set of actions;*
- $\rightarrow \subseteq S^a \times A \times S$ *is an action transition relation ;*
- $\sigma : S^p \rightarrow (S \rightarrow 2^{[}0,1])$ *is a transition probability predicate such that for each pair* $(s, s') \in S^p \times S$*,* $\sigma(s)(s')$ *is an interval of probabilities.*
- $s_0 \in S$ *is the initial state.*

This contract definition is an extension of the probabilistic specification defined in Section 5.3.1. A transition $(s, a, \bot)$ in a contract is an assumption that action a does not occur in s, whereas a transition (s, a, s') with $s' \neq \bot$ is a guarantee that action a will be accepted in s. If as sate s has not outgoing transitions labelled with action a, than the component guarantees that such actions will not be emitted at s. A transition (s, s') such that $\sigma(s)(s')$ defines the allowed transition probabilities from s to s'.

In this work, the authors define composition and conjunction of contracts that do not seem to present any difficulty. The authors also define contract refinement that for the probabilistic transitions follows the notion of simulation defined in Section 5.3.1. In summary, given two contracts $C_1(S_1, A_1, \rightarrow_1, \sigma_1, s_{0,1})$, and $C_2(S_2, A_2, \rightarrow_2, \sigma_2, s_{0,2})$, C_1 refines C_2 if and only if $s_{0,1} \leq s_{0,2}$ where $\leq \subseteq S_1 \times S_2$ is the greatest relation such that $s \leq t$ implies the following:

- C_1 does not make stronger assumptions than C_2, i.e. $s = \bot \Rightarrow t = \bot$

- C_1 must provide at least the same guarantees, meaning that if there is a transition (t, a, t') $(t' \neq \bot)$ in C_2, then there must be a corresponding transition (s, a, s') in C_1 and $s' \leq t'$.
- If s and t are probabilistic states, then t must simulate s (according to the definition in Section 5.3.1).
- If s is an action (probabilistic) state and t is a probabilistic (action) state, then s (all actions states reachable from s with paths of non-zero probability) must refine all action states reachable from t over paths of non-zero probability (t).

The verification of contract refinement entails finding the greatest relation $\leq$ which is possible, but it might face complexity issues for very large models (e.g. the ones obtained through discretization of stochastic hybrid systems).

PCBD using Markov Decision Processes Delahaye [23] proposes to use a model that can be casted to a Markov Decision Process. This model is the used to compute the probability that an implementation steps outside of a contract specification using the notion of stationary optimal strategies with mean-payoff functions [32].

The main idea can be described as follows. Consider a Controllable Markov Chain (CMC), that is a tuple (S, A, α, P) where S is a set of states, A is a set of actions, $\alpha : S \to 2^A$ is a function that associates to each state a set of available actions, and $P : S \times A \to (S \to [0, 1])$ a transition probability function that associates to a state s and an action a a probability distribution over S. A pure stationary strategy for a CMC is a function $\sigma : S \to A$ that selects an action in each state of the CMC.

A Markov Decision Process (MDP) is a CMC with an associated payoff function ϕ. Given an MDP, one can define a probability measure on the execution paths induced by a pure strategy σ (and it can be proved that such probability measure is unique [32]). Let $\mathbb{P}_s^\sigma$ denote such probability measure of the path starting at s. A payoff function is a function $\phi : S^\omega \to \mathbb{R}$ that maps infinite sequences of states to a payoff. Then, the expected payoff is $E_s^\sigma(\phi(S_0 S_1 \ldots))$.

In this work, the payoff function is define as follows. A reward $r(s, a, t)$ is associated with each transition from s to t under action a. The mean payoff is then defined as:

$$\limsup_{n \in \mathbb{N}} \frac{1}{n+1} \sum_{i=0}^{n} r(s_i, a_i, s_{i+1})$$

In [32] it is proved that mean-payoff MDPs have a pure stationary optimal strategy.

In this work, models are deterministic but they may have probabilistic inputs. A contract is then defined as follows:

Definition 18. *A probabilistic contract is a tuple $C(\mathbf{u}, \mathbf{c}, \mathbf{p}, \mathbb{D}, E)$ where*

- $\mathbf{u} \cup \mathbf{c}$ *is the signature of the contract, where* $\mathbf{u}$ *is the set of uncontrolled variables and* $\mathbf{c}$ *is the set of controlled variables*
- $\mathbf{p} \subseteq \mathbf{u}$ *is the set of probabilistic inputs*
- $\mathbb{D} : \mathbf{p} \to Dist(V)$ *is a function that associates to each probabilistic port a probability distribution on the domain of values V of the ports.*
- $E(S, A, \delta)$ *is a deterministic transition system.*

A contract is the used to derive a CMC E_p which is obtained as a result of having probabilistic inputs to a deterministic transition system. Given an implementation $M(S_M, A, \delta_M)$, the synchronous product of E_p and M is used to define a MDP where the reward structure is defined as follows. Let (s_1, s_2) be a state of the product of E_p and M, respectively, then:

- If an action a is available in s_1 and in M, then
$$r((s_1, s_2), a, (s_1', s_2')) = 0$$
- If an action a is not available in s_1 and it is available in s_2, then
$$r((s_1, s_2), a, (s_1, s_2')) = 1$$
- if an action a is available in s_1 and $\delta_M(s_2, a)$ is not defined, then
$$r((s_1, s_2), a, (s_1', s_2)) = 0$$

For this MDP, one can find a pure strategy that provide the maximal expected mean-payoff β. This value can be seen as the maximal expected probability that an implementation violates the contract. Thus the model M satisfies contract C with probability at least $1 - \beta$.

Assume/Guarantee reasoning The third approach we present has been recently implemented in the PRISM [40] probabilistic model checker. In this approach, assumes and guarantees about a model are captured by safety property. Given a safety property A, a deterministic finite automata that define all the prefixes of the model's behaviors that violate the property is defined. A deterministic finite automaton is a tuple $A^{err}(S, s_0, \alpha_A, \delta_A, F)$ where S is the set of states, $s_0 \in S$ is the initial state, α_A is an alphabet (of actions), $\delta_A : S \times \alpha_A \to S$ is a transition function and $F \subseteq S$ is a set of accepting (error) states (labelled with err_A).

Checking if a probabilistic automaton M satisfies a safety property with a certain minimum probability, written $M \models \langle A \rangle_{\geq p}$ requires the following two steps:

- Derive a probabilistic automaton $M' = M \otimes A^{err}$ (this operation is defined by the authors in [40]).
- Compute $Pr_M^{min}(A) = 1 - Pr_{M \otimes A^{err}}^{max}(\diamond err_A)$, i.e. the minimum probability that M satisfies A
- Compare this probability with p.

In assume guarantee reasoning, one wants to check the following basic property:

$$\langle A \rangle_{\geq p_A} M \langle G \rangle_{\geq p_G}$$

meaning that if the assumption A is verified with probability at least p_A, then M satisfies the guarantee G with probability at least p_G. This basic statement is used in inference rules like the following:

$$\frac{\langle true \rangle M \langle A \rangle_{\geq p_A} \wedge \langle A \rangle_{\geq p_A} M \langle A \rangle_{\geq p_G}}{\langle true \rangle M \langle G \rangle_{\geq p_G}}$$

In [40] it is shown that checking:

$$\langle A \rangle_{\geq p_A} M \langle G \rangle_{\geq p_G}$$

reduces to checking the following:

$$\neg \exists \sigma' \in Adv_{M'} \ s.t. \ \left(Pr_{M'}^{\sigma'}(\Box \neg err_A) \geq p_A \wedge Pr_{M'}^{\sigma'}(\diamond err_G) > 1 - p_G \right)$$

This statement can be informally understood as follows. Consider the (possibly non-deterministic) model $M' = M[\alpha_A] \otimes A^{err} \otimes G^{err}$ (i.e. the model composed with the probabilistic automata for the assumptions and guarantees, respectively. Then one needs to check that there is no strategy σ' that can generate a trace of the model M' with a probability of not getting into an assumption error state greater than p_A (i.e. that satisfies the assumption), and with a probability of getting into a guarantee error state greater than $1 - p_G$ (i.e. that violates the guarantee).

This checking can be done in polynomial time using multi-objective model checking [27].

5.4 Approximate Refinement Checking

In the final remarks of Section 5.2, we emphasize the complexity barrier that is faced when checking for contract refinement. In this section we first introduce a probabilistic model with inputs and outputs that will be used to define refinement of models. We then use the definitions given in Section 5.3 to find the conditions under which two models can be claimed to be in refinement relation. Finally, we propose the use of an approximate refinement checking which provides a distance between two models. A small distance is an indication that the two model are close to be one the refinement of the other.

5.4.1 Stochastic Input-Output Automata

The Stochastic Input-Output Automata (SIOA) model (presented later in this section) is a simple model that we use to describe the specification of a system and its refinement.

Definition 19. *A stochastic input-output automaton is a collection* $IOA = \{I, S, O, T, U\}$*, where*

- I *is a finite discrete space of inputs.*
- S *is a finite discrete space of internal states for the IOA.*
- O *is a finite discrete space of outputs.*
- T *gives the transition probabilities for the internal state of the IOA given the current value of the input. If* $s \in S$ *and* $i \in I$*, then* $T(.,(s,i))$ *is a probability distribution on the internal state space* S*. i.e.,* $T(s',(s,i))$ *is the probability for the internal state of the IOA to transition from* s *to* s' *given that the current value of the input is* i*.*
- U *gives the conditional probability for the outputs given the current values for the internal state and input.* $U(.,(s,i))$ *is a probability distribution on the output space* O*. i.e.,* $U(o,(s,i))$ *gives the probability for the output to be* o *given that the current values for the internal state and input are* s *and* i *respectively.*

The execution of an SIOA is defined as follows. For a given input sequence $\{i_1, i_2, ..., i_k, ...\}$, the input-output automaton executes as follows. At a given time k, if the internal state is s_k, the output o_k is generated randomly by extracting a value from the distribution $U(.,(s_k,i_k))$. The internal state at the next time-step s_{k+1} is chosen by extracting a value from the distribution $T(.,(s_k,i_k))$. This procedure is repeated at every time-step.

5.4.2 Equivalence of SIOAs

In this section we define when two SIOAs are equivalent. We then introduce a metric that measures the degree to which two SIOAs are equivalent and that can be used to judge whether an implementation can be accepted as refinement of a specification.

Let $P : S_r \to S_c$ be a mapping. In our discussions, S_r is going to refer to a larger (or refined) state-space and S_c to a smaller (or coarse) state-space. The mapping P can be thought of as a projection. Given a mapping P, we also define a corresponding mapping $P^e : S_r \times I \to S_c \times I$ defined as $P^e(s_r, i) = (P(s_r), i)$. Let M_r be the space of probability distributions on the space S_r and M_c be the space of probability distributions on the space S_c. We define a mapping $Pr : M_r \to M_c$ as follows:

$$Pr(\mu_r)(s_c) = \sum_{s_r \in S_R : P(s_r) = s_c} \mu(s_r). \tag{5.2}$$

where $\mu_r \in M_r$ and $Pr(\mu_r) \in M_c$. We refer to Pr as the *probability projection mapping* corresponding to the projection P.

In the following discussions, it would be useful to define the following mappings $V_i : M \to M_o$ where i is a member of the input space I and M_o is the space of probability distributions on the output space O. V_i is defined in terms of the conditional probability U as:

$$V_i(\mu)(o) = \sum_{s \in S} \mu(s) U(o, (s, i)). \tag{5.3}$$

where $\mu \in M$ and $V_i(\mu) \in M_o$. We refer to V_i as the *state-output projections* corresponding to U.

We now define *refined implementations* of stochastic input-output automata.

Definition 20. *A stochastic input-output automaton $IOA_r = \{I, S_r, O, T_r, U_r\}$ is said to be a refined implementation of the stochastic input-output automaton $IOA_c = \{I, S_c, O, T_c, U_c\}$ if there exists a projection mapping $P : S_r \to S_c$ that satisfies the following conditions.*

1 For all pairs of values $(s_r, i) \in S_r \times I$, we must have $Pr(T_r(., (s_r, i))) = T_c(., (P(s_r), i))$.

2 For all pairs of values $(s_r, i) \in S_r \times I$, we must also have $U_r(., (s_r, i)) = U_c(., (P(s_r), i))$.

The first condition in Definition 20 requires that the one-step transition of the refined internal state followed by the projection should give the same result as projecting the refined state followed by the one-step transition of the coarse internal state. The commutative diagram in Figure 5.2 illustrates this. The second condition in Definition 20 requires that the outputs of the refined system corresponding to the internal state s_r must be the same as the outputs of the coarse system corresponding to the projected state $P(s_r)$.

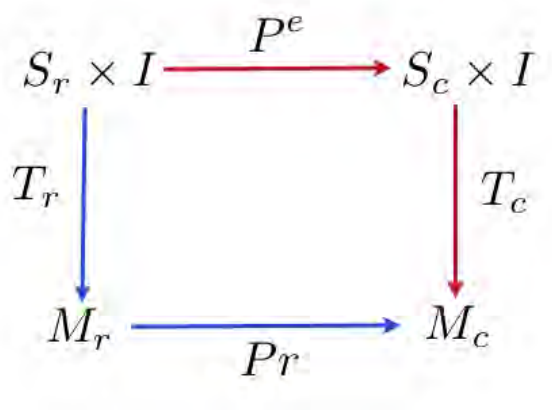


Figure 5.2: Commutative diagram illustrating the first condition in Definition 20

Theorem 5.4.1. *If $IOA_r = \{I, S_r, O, T_r, U_r\}$ is a refined implementation of $IOA_c = \{I, S_c, O, T_c, U_c\}$, then they have the same input-output behavior. Assuming that the initial probability distributions for the states of the two systems are consistent ($\mu_0^c = Pr(\mu_0^r)$), then for a given input sequence $\{i_1, ..., i_N\} \in I^N$, any output sequence $\{o_1, ..., o_N\} \in O^N$ has the same probability of occurring under executions of IOA_r and IOA_c.*

Proof. Let W_i^r and W_i^c respectively be the transition matrices for the states of the refined system and the coarse system corresponding to an arbitrary input i. Let μ^r and μ^c be probability distributions

for the states of the refined and coarse system such that $\mu^c = Pr(\mu^r)$. Let V_i^r and V_i^c be the *state-output projection* mappings corresponding to U_r and U_c respectivly. All we need to prove the above Theorem is to show that

$$V_{i'}^r(\mu^r W_i^r) = V_{i'}^c(\mu^c W_i^c) \text{ for all } i, i' \in I. \tag{5.4}$$

The statement above essentially states that the sequence of inputs i and i' lead to the same probability distribution of the outputs for the both the coarse and refined system. To prove the above statement, first observe that

$$Pr(\mu^r W_i^r) = Pr(\mu^r) W_i^c = \mu^c W_i^c. \tag{5.5}$$

This is because we have

$$\begin{aligned} L.H.S = Pr(\mu^r W_i^r) &= Pr\left(\left(\sum_{s_r} \mu^r(s_r)\delta_{s_r}\right) W_i^r\right) \\ &= Pr\left(\sum_{s_r} \mu^r(s_r)\delta_{s_r} W_i^r\right) \\ &= Pr\left(\sum_{s_r} \mu^r(s_r) W_i^r(s_r, .)\right) \\ &= \sum_{s_r} \mu^r(s_r) Pr\left(W_i^r(s_r, .)\right) \\ &= \sum_{s_r} \mu^r(s_r) W_i^c(P(s_r), .). \end{aligned} \tag{5.6}$$

The last equality follows from the commutative requirement in Condition 1 of Definition 20. We also have

$$\begin{aligned} R.H.S = \mu^c W_i^c &= Pr(\mu^r) W_i^c \\ &= Pr\left(\sum_{s_r} \mu^r(s_r)\delta_{s_r}\right) W_i^c \\ &= \left(\sum_{s_r} \mu^r(s_r) Pr(\delta_{s_r})\right) W_i^c \\ &= \left(\sum_{s_r} \mu^r(s_r)\delta_{P(s_r)}\right) W_i^c \\ &= \sum_{s_r} \mu^r(s_r) W_i^c(P(s_r), .). \end{aligned} \tag{5.7}$$

Thus the L.H.S and R.H.S are equal. In the above expressions, δ_{s_r} is a probability distribution such that

$$\delta_{s_r}(s) = \begin{cases} 1 \text{ if } s = s_r \\ 0 \text{ if } s \neq s_r. \end{cases} \tag{5.8}$$

Now, from Condition 2 of Definition 20, we have

$$V_{i'}^r(\mu^r W_i^r) = V_{i'}^c(Pr(\mu^r W_i^r)) = V_{i'}^c(\mu^c W_i^c). \tag{5.9}$$

This completes the proof as we can now easily extend the argument for an arbitrary length of input sequences as we always have $\mu_k^c = Pr(\mu_k^r)$ where μ_k^r and μ_k^c are the probability distributions for the refined and coarse states at time-step k. □

5.4.3 Metrics for comparing SIOAs

Given stochastic input-output automata $IOA_r = \{I, S_r, O, T_r, U_r\}$ and $IOA_c = \{I, S_c, O, T_c, U_c\}$, ideally one would like to check for refinement by checking for Conditions 1 and 2 in Definition 20 for all $(s_r, i) \in S_r \times I$. This naturally leads to the following metric:

$$\begin{aligned} Dist(IOA_r, IOA_c) = &\sum_{i \in I} \sum_{s_r \in S_r} |Pr(T_r(., (s_r, i))) - T_c(., (P(s_r), i))| \\ &+ \sum_{i \in I} \sum_{s_r \in S_r} |U_r(., (s_r, i)) - U_c(., (P(s_r), i))| . \end{aligned} \tag{5.10}$$

Computing the above metric may be infeasible and unnecessary because when these automata are connected to other automata, many internal states may never be reached and the set of inputs with which the automata may be excited may be constrained due to the dynamics of the composed system. Given a probability distribution $\bar{\mu}^I$ on the space of inputs and a probability distribution $\bar{\mu}^r$ on the refined internal states, we could define the following metric:

$$\begin{aligned} Dist_{\mu^i, \mu^r}(IOA_r, IOA_c) = &\sum_{i \in I} \sum_{s_r \in S_r} \bar{\mu}^I(i) \bar{\mu}^r(s_r) |Pr(T_r(., (s_r, i))) - T_c(., (P(s_r), i))| \\ &+ \sum_{i \in I} \sum_{s_r \in S_r} \bar{\mu}^I(i) \bar{\mu}^r(s_r) |U_r(., (s_r, i)) - U_c(., (P(s_r), i))| . \end{aligned} \tag{5.11}$$

$\bar{\mu}^I(i)$ could reflect the probability that the input i will be received and $\bar{\mu}^r(s_r)$ could reflect the probability that the state s_r will be touched. Thus inputs that are never received and internal states that are never reached do not influence the metric. Therefore the refinement of an automaton can be checked in the context determined by the assumptions on its environment. A natural choice for $\bar{\mu}^I(i)$ and $\bar{\mu}^r(s_r)$ will be

$$\begin{aligned} \bar{\mu}^I(i) &= \frac{1}{T} \int_0^T \mu(i, t) dt \\ \bar{\mu}^r(s_r) &= \frac{1}{T} \int_0^T \mu(s_r, t) dt. \end{aligned} \tag{5.12}$$

where $\mu(i, t)$ is the probability of the input being i at time t and $\mu(s_r, t)$ is the probability of the internal state being s_r at time t.

Chapter 6

Refinement checking applied to the heat load component

The system presented is Chapter 1 is refined into more detailed models for the heat load and the heat sink. We refine the heat load sub-system by including the details of the oil circuit used to cool down the generators and the engine. We then show how refinement checking can be carried out between the original lumped model and the more refined model.

6.1 Refined model description

We refine the heat load component of the thermal management system. In particular we are interested in the oil circuit that is used to cool down the engine or the electric power generators. Figure 6.1 shows a block diagram of the oil circuit used to reject the heat from these type of sources. Consider for example the heat from a power generator (we consider the generator only in this section as the same method is used to transfer heat from the engine to the fuel). The generator needs to meet a power requirement w. It generates a certain amount of heat H_G that needs to be rejected. The heat is transferred to the oil that is typically used as lubricant. The oil circuit is very similar to the fuel circuit presented in Chapter 1.

The oil is collected in a tank that is modeled as a lumped element containing a certain oil mass M_o at temperature T_o. Contrary to the fuel circuit, the total oil mass M_o is constant over time (unless there are leaks in the circuit). The oil is moved by a pump that is controlled by a controller which reads the oil temperature and acts on the oil mass rate $\dot{m}_o$. The heat transferred to the oil is then transferred to the fuel through an oil/fuel heat exchanger. The heat exchanger has two sides: the oil side with an oil inlet and an oil outlet, and a fuel side with a fuel inlet and a fuel outlet. The dashed line box represents the model in Chapter 1, Figure 1.4. This refined model is supposed to be substituted to the heat load component of Figure 1.4.

A naïve approach to the verification of the refined model is to simply substitute the heat load component with its refinement. This substitution increases the number of state variables which makes the complexity of the verification task unmanageable. The use of contract-based design avoids this complexity by proceeding as follows. Assume the heat load component to be modeled as a contract with assumptions on the inputs (i.e. heat load H_L, the fuel flow rate $\dot{m}_{out}$, and the temperature T_out of the fuel at the inlet of the heat load component). Under these assumptions, the heat load component provides as a guarantee a step change in the temperature of the fuel while leaving the flow rate unchanged[1].

[1]This guarantee is hard to meet for any realistic implementation and will have to be relaxed in the refinement process when enough details are available. Alternatively, the guarantee could be already changed at this level by

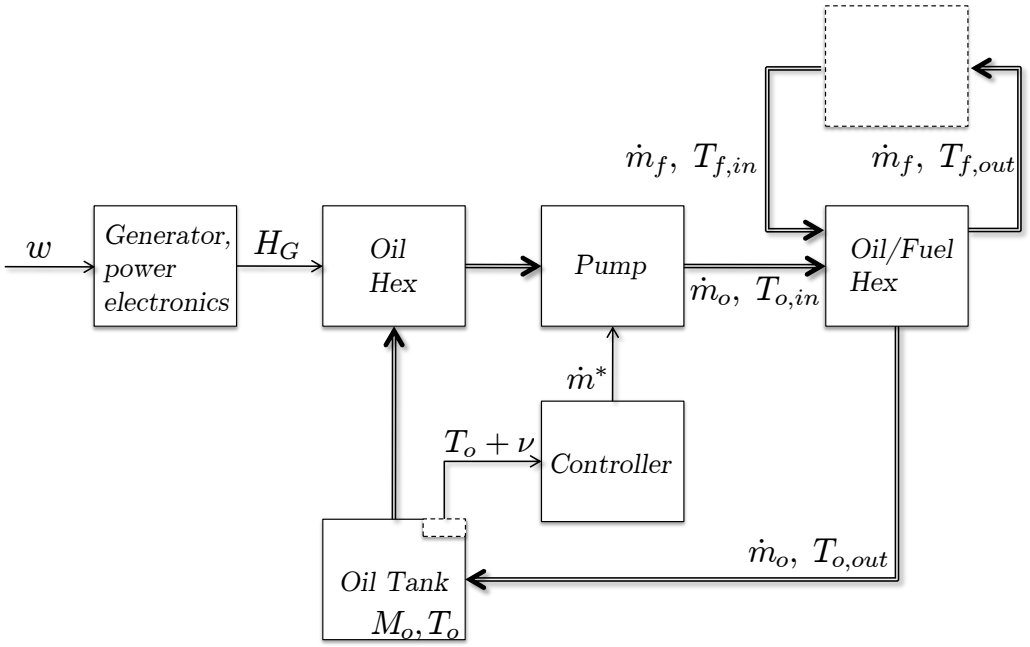


Figure 6.1: Refinement of the oil circuit on the electric power system side.

There are challenges associated with the ability to extract assumptions when they are not given. To be more precise, our model did not include an explicit assumption on the fuel flow rate and on fuel temperature. The only assumption is on the heat load which we assumed to be uniformly distributed around a mean value given for each mission point. The assumptions on the input temperature and on the fuel flow rate should in principle be derived by the analysis of the abstract model. It is beneficial to be able to derive tight assumptions to avoid flowing down requirements that would be, otherwise, too stringent.

Section 5.3 provides a review of methods that have been used to define probabilistic contracts. One approach to derive assumptions for our heat load component is to select a modeling paradigm (e.g. Interactive Markov Chains), construct a parametric model, and estimate the parameters of the model through simulation, or again analysis. Another approach is simply to perform analysis of the abstract model and record the evolution of the probability distribution functions at the input of the heat load component. These probability distributions can then be used to compare the abstract and the refined model as done later in this section.

6.1.1 Refined model for fuel-oil heat exchanger

The effectiveness of a heat exchanger is defined as the ratio between the actual heat transferred and the maximum heat transfer possible with an ideal heat exchanger. Consider the fuel-oil heat exchanger shown below.

The effectiveness is defined as

introducing a slow dynamics in the temperature change and a probability distribution around it which captures the uncertainty due to abstraction.

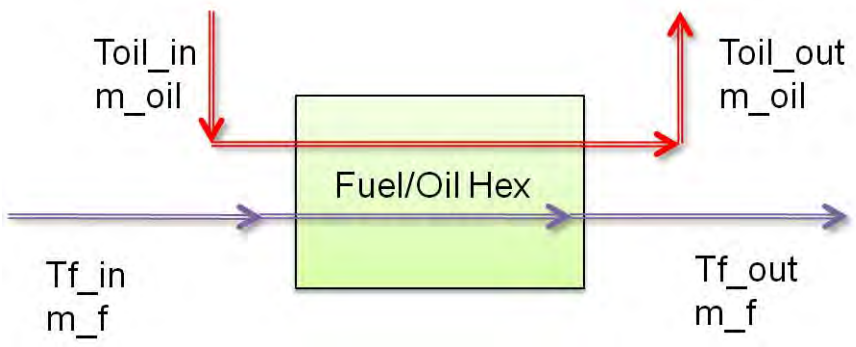


Figure 6.2: Fuel-oil heat exchanger

$$\begin{aligned} \epsilon_{f/o} &= \frac{m_f c_f \left(Tf_{out} - Tf_{in}\right)}{m_{oil} c_{oil} \left(Toil_{in} - Tf_{in}\right)} \\ &= \frac{m_{oil} c_{oil} \left(Toil_{in} - Toil_{out}\right)}{m_{oil} c_{oil} \left(Toil_{in} - Tf_{in}\right)}. \end{aligned} \tag{6.1}$$

The above equations are written assuming that $(m_{oil}c_{oil})$ represents the minimum thermal-capacity rate. The abstract model of the heat load used in Chapter 1 *guarantees* that the heat H_L is entirely transferred to the fuel. This guarantee must also be provided by the refined model. The guarantee can be satisfied by controlling the oil flow rate as follows:

$$m_{oil} = \frac{H_L}{\epsilon_{f/o} c_{oil} \left(Toil_{in} - Tf_{in}\right)}. \tag{6.2}$$

A controller that achieves the above oil flow rate satisfies the contract that a heat load H_L is dumped into the fuel. There are two reasons why the controller may fail to satisfy the guarantee. The first reason is that the dynamics of the controller might be slow to adapt to abrupt changes in the heat H_L. The second reason is that temperature measurements might be affected by noise (or imprecision). We consider the second effect and we show how the approximate refinement relation changes depending on the level of the noise assumed on the measurement.

6.2 Refinement checking results

In this section we start by comparing two simple models: The high level model of the system where which uses an ideal heat load component, and a more refined model where the heat is exchanged between through a fuel-oil heat exchanger (as described in Section 6.1.1). In this experimental setup we proceed as shown in Figure 6.3. We build two heat exchanger models:

- An abstract model, called *coarse* which follows the implementation presented in Section 1.1. In this model, the heat that is transferred to the fuel induces just a change in the fuel temperature which is proportional to the heat load divided by the fuel rate.
- A *refined* model which follows the implementation presented in Section 6.1.1. In this model, the effect of the oil circuit is taken into account. In particular, the total oil mass is assumed to be 850 kg.

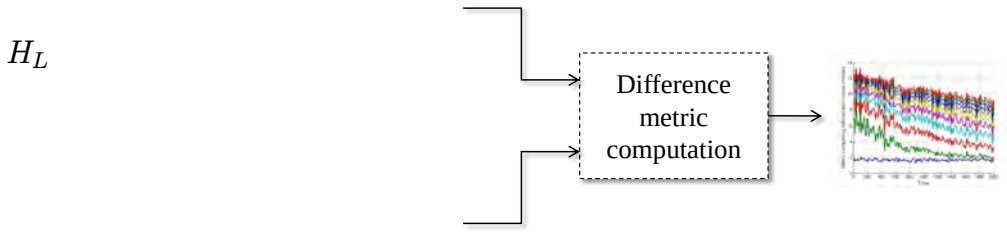


Figure 6.3: First experimental result for the refinement verification of the heat exchanger.

The inputs to the models are the total heat load H_L, the fuel rate $\dot{m}_f$ and the fuel temperature T_f. The output of the model is the oil temperature. We generate a probability distribution over the input space. In this case study, the probability distribution is supposed to be uniform between the following bounds: $H_L \in [20, 25]$ kW, $\dot{m}_f \in [1, 2] kg/s$, and $T_f \in [280, 300]$ K. This distribution is sampled at each time step of the reachability analysis algorithm to generate the distributions over the next states and over the outputs. The distance metric on the outputs has been defined in Section 5.4.3. We plot the result in Figure 6.4. The metric is plotted as a function of time. For a fixed time, higher values of the metric correspond to higher variance of the sensor noise (as expected)

Figure 6.5 shows the time average of the distance metric as a function of the sensor noise variance.

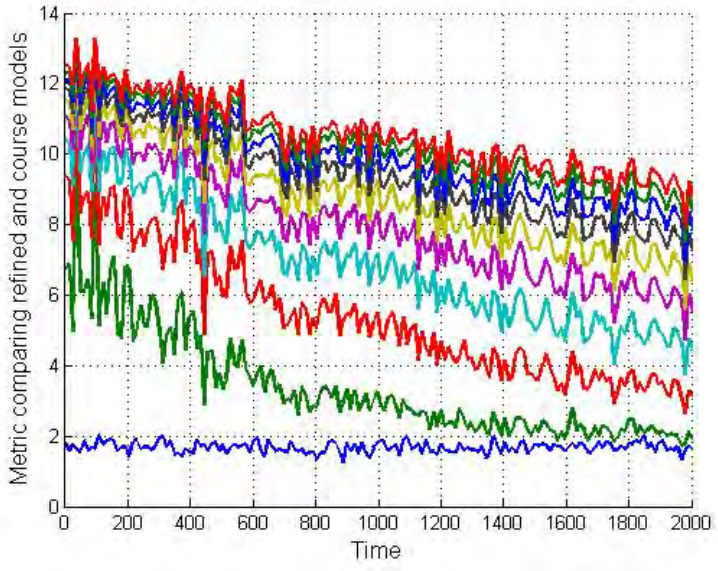


Figure 6.4: Value of the output distance between the coarse and refined model of the heat exchanger for different values of the variance of the sensor noise.

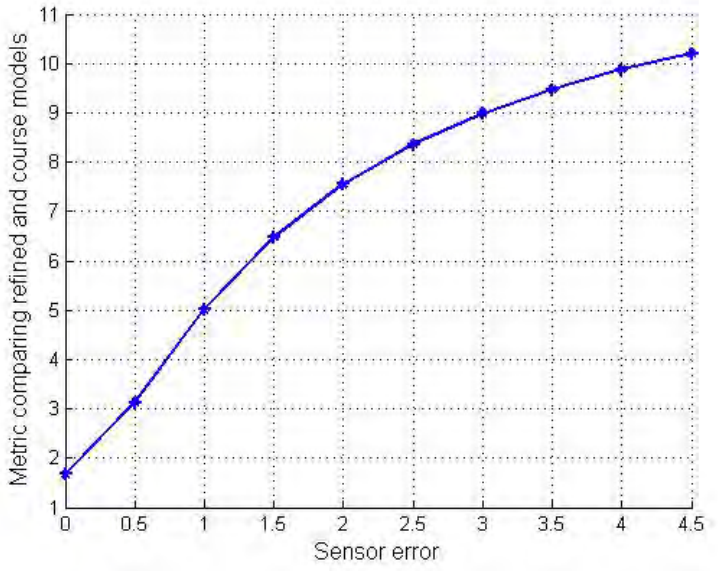


Figure 6.5: Time average of the distance between the two models as a function of the variance of the sensor noise.

Chapter 7

Optimal control of the fuel temperature

In this chapter and the ones that follow, we show how one could move away from the complexity of the analysis problem by *synthesizing* systems in such a way that the desired properties are verified *by construction*. In other words, instead of verifying that a property P is satisfied for a given system, we aim at developing tools such that the system is the result of an automated design process which takes P as constraint (and therefore generates only systems that satisfy such property).

In this chapter, we show an optimal control synthesis technique for the thermal management system. In this approach, the natural dynamics of the system is considered given while the control laws are considered parameters to be optimized. Bounds on the fuel temperature are taken into account as constraints of an optimization problem. Thus, the required guarantees that the system must provide are enforced by construction, i.e. through a synthesis procedure, such that verification is not needed. We take into account the stochastic nature of the dynamics of the plant. In Chapter 9 we generalize the synthesis approach to include the effect of the implementation platform such as the reliability of hardware components.

For plants that can be modeled as hybrid systems, we may also need a controller that is hybrid in nature. Design of such hybrid controllers or computation of optimal controls for hybrid systems in general is a difficult task. For some results, see [34] and [21]. In [34], the authors present a method to compute approximations to optimal feedback control laws using discretizations of Bellman type inequalities for lower bounds on the optimal value function. In [21], the authors discuss necessary conditions for optimality for a class of hybrid systems and show how this leads to non-smooth optimization problems. However, efficient algorithms for solving these non-smooth optimization problems are still not available. In [18], the author discusses algorithms which efficiently solve control synthesis problems for constrained linear systems and constrained linear hybrid systems. There are few or no tools currently available for optimal control of nonlinear hybrid systems or stochastic hybrid systems.

In this chapter, we focus on numerical methods for a class of nonlinear hybrid systems where the mode transitions are enabled by a clock value. i.e., mode transitions are dependent only on the amount of time spent in a mode and not on the actual state of the system. The amount of time spent in a mode of operation (the transition time) is also assumed to be a random variable with a known distribution. Thus the transition times can be thought of as uncertain parameters of the hybrid system. Note that these transitions times are not part of the design and are a characteristic of the system to be controlled. The hybrid system also has some free adjustable parameters corresponding to each mode of operation that can be chosen by the designer. The objective is to pick values for these adjustable parameters of the system that minimizes the expected value of some meaningful cost-function that captures the behavior of the system over a finite time-horizon. Thus the cost-function

may include finite-time-integrals of some function of the states of the system and the controls.

We show an optimal control synthesis technique to minimize the expected value of a cost-function. This can be framed as a stochastic optimization problem. We use the *sample average approximation* method described in [39] to solve the stochastic optimization problem. The basic idea behind the *sample average approximation* method is simple. It is a Monte Carlo sampling-based approach to stochastic optimization problems. A random sample of the uncertain parameters is generated and the expected value function is approximated by the corresponding sample average function. The resulting sample average optimization problem is solved using standard optimization techniques. We use the optimization software IPOPT ([52]) to numerically solve the resulting sample average optimization problem.

As a case study, we illustrate the method for the design of a thermal management system of a prototypical aircraft. The parameters to be optimized for are the fuel flow rates of the system during various phases of the mission and the uncertain parameters are the time durations of various phases of the mission. In principle it would also be possible to take into account the effects due to the implementation platform such as delays in computation and communication, and reliability of communication.

7.1 Optimal design of hybrid systems with time-triggered mode transitions

In this section, we discuss an approach to compute optimal controls for discrete-time hybrid systems with uncertain parameters. In particular, we look at discrete-time hybrid systems with time-triggered mode transitions. i.e., the switching of the system from one mode to another depends only on the amount of time spent in that mode (or equivalently a clock value). The amount of time spent in each mode is a random variable with some known distribution. Note that for such systems, the sequence of modes that the system operates in is known beforehand. Thus the system we consider has dynamics of the form

$$\begin{aligned} x(k+1) &= T(1, x(k), p_0) \text{ for } j_0 \leq k < j_1 \\ x(k+1) &= T(2, x(k), p_1) \text{ for } j_1 \leq k < j_2 \\ x(k+1) &= T(3, x(k), p_2) \text{ for } j_2 \leq k < j_3 \\ &\ldots\ldots \\ x(k+1) &= T(m, x(k), p_m) \text{ for } j_{m-1} \leq k < j_m. \end{aligned} \tag{7.1}$$

Here k represents the time-index. The amount of time spent in each mode $\Delta_q = j_q - j_{q-1}$, is a random variable with some known distribution $\mathcal{W}_q$. The system has m modes and the $m-th$ is considered to be the 'final' mode. The time duration of a sample trajectory is given as

$$L = \sum_{q=1}^{m} \Delta_q. \tag{7.2}$$

Systems described above are particularly useful to model processes where the sequence of modes of operation are fixed and known in advance, but the transition times are uncertain. A good example for such a system is the typical mission of an aircraft which has various modes of operation like *Taxing*, *Take-off*, *Flying* and *Landing*. A typical aircraft mission follows a fixed sequence of modes, but the time spent in modes like *Taxing* and *Flying* may be uncertain.

The variables p_q are parameters that need to be optimized for in the operation of each mode. In what follows, we describe how one can do such an optimization. The cost-function we use is

the expected value of some functional of the sample trajectories. The optimization problem can be written as

$$\min_{p_q \in U_q} \left\{ C := \mathbb{E}_\Delta \left[\sum_{q=1}^{m} \sum_{k=j_{q-1}}^{j_q} F_q\left(x(k), p_q\right) \right] \right\}. \tag{7.3}$$

where $x(k)$ is subject to the dynamics described in (7.1). F_q is assumed to be a differentiable function of $x(k)$ and p_q. Δ is the vector of transition times. i.e. Δ_q is the $q-th$ component of the vector Δ. The expectation is taken over the uncertain transition times Δ_q. To solve the stochastic optimization problem described above, the *sample average approximation* (SAA) method (see [39]) is a natural choice. We briefly review the SAA method in the following subsection.

7.1.1 Review of the sample average approximation method

In the most general form, stochastic optimization problems take the form

$$\min_{u \in U} \left\{ g(u) := \mathbb{E}_P G(u, W) \right\}. \tag{7.4}$$

Here W is a random vector having probability distribution P. U is the set from which the variables u can be chosen from. For the optimal design problem we described before, the random vector W would correspond to the vector Δ whose elements are the random transition times for each mode. The variables u include both the parameters p_q that need to be optimized for and the states $x(k)$ that are subject to the constraints imposed by the system dynamics. $\mathbb{E}_P G(u, W) = \int G(u, W) P(dw)$ is the expected value of the objective function $G(u, W)$. The SAA method is suitable for optimization problems that have the following characteristics.

- The expected value function $g(u) := \mathbb{E}_P G(u, W)$ cannot be written in a closed form and its value cannot be easily calculated.
- The function $G(u, W)$ is easily computable for given u and W.
- The set of feasible solutions U is very large so that enumeration is not feasible.

The optimal design problem we described before has all these characteristics and therefore makes the *sample average approximation* method a natural approach to this problem. The basic idea of *sample average approximation* is simple indeed. It is a Monte Carlo sampling-based approach to stochastic optimization problems. A random sample of W is generated and the expected value function is approximated by the corresponding sample average function. The obtained sample average optimization problem is solved, and the procedure is repeated several times until a stopping criterion is satisfied.

Let $W^1, ..., W^N$ be an independently and identically distributed (i.i.d) random sample of N realizations of the random vector W. Consider the sample average function

$$\bar{g}_N(u) := \frac{1}{N} \sum_{i=1}^{N} G(u, W^i). \tag{7.5}$$

The sample average approximation (SAA) problem is

$$\min_{u \in U} \bar{g}_N(u). \tag{7.6}$$

It has been shown that the solution of the sample average approximation problem (7.6) converges to the solution of the original problem (7.4) with probability one. Also roughly speaking, the optimal value for the objective function obtained from the approximate problem converges exponentially fast to the true optimal value for the objective function as $N \to \infty$. For more theoretical details on the convergence of the SAA method, see [39].

7.1.2 Application of SAA to optimization of time-triggered hybrid systems

For hybrid systems of the type described in (7.1) at the beginning of this chapter, optimal design can be done using the SAA method described before. One can generate a finite number of samples for the random vector of transition times Δ and then find the optimal parameters p_q that minimize the corresponding sample average function. The variables u that need to be optimized for include both the state variables x_k^i corresponding to each sample Δ^i and the parameters p_q. In our notation, Δ^i for $i = 1, 2, ..., N$ are N i.i.d. samples for the vector of transitions times and x_k^i is the state of the system at time k for a trajectory corresponding to the sample Δ^i. The state variables x_k^i are subject to the constraints imposed by the system dynamics given as

$$\begin{aligned} g_k^i &= x_{k+1}^i - T(1, x_k^i, p_0) = 0 \text{ for } j_0 \leq k < j_1^i \\ g_k^i &= x_{k+1}^i - T(2, x_k^i, p_1) = 0 \text{ for } j_1^i \leq k < j_2^i \\ g_k^i &= x_{k+1}^i - T(3, x_k^i, p_2) = 0 \text{ for } j_2^i \leq k < j_3^i \\ &..... \\ g_k^i &= x_{k+1}^i - T(m, x_k^i, p_m) = 0 \text{ for } j_{m-1}^i \leq k < j_m^i. \end{aligned} \tag{7.7}$$

where $j_q^i - j_{q-1}^i = \Delta_q^i$. The cost-function is approximated by the sample average given as

$$\bar{C} = \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{q=1}^{m} \sum_{k=j_{q-1}^i}^{j_q^i} F_q \left(x_k^i, p_q \right) \right]. \tag{7.8}$$

To find the state variables x_k^i and the parameters p_q that minimizes the cost-function $\bar{C}$ subject to the constraints in (7.7), we use the optimization software IPOPT (see [52]). IPOPT is a software package for large-scale nonlinear optimization. It uses an interior point line search filter method to find local solutions to nonlinear optimization problems. For the IPOPT software, it is necessary to compute the gradient of the cost-function $\bar{C}$ with respect to the variables x_k^i and p_q. These derivatives are given as

$$\frac{\partial \bar{C}}{\partial x_k^i} = \frac{1}{N} \frac{\partial F_q}{\partial x_k^i} \tag{7.9}$$

$$\frac{\partial \bar{C}}{\partial p_q} = \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{k=j_{q-1}^i}^{j_q^i} \frac{\partial F_q}{\partial p_q} \right]. \tag{7.10}$$

It is also necessary to compute the Jacobian of the constraint equations. The elements of the Jacobian of the constraint equations are computed as

$$\begin{aligned} \frac{\partial g_k^i}{\partial x_{k+1}^i} &= 1.0 \\ \frac{\partial g_k^i}{\partial x_k^i} &= \frac{-\partial T(q, ., .)}{\partial x_k^i} \\ \frac{\partial g_k^i}{\partial p_q} &= \frac{-\partial T(q, ., .)}{\partial p_q}. \end{aligned} \tag{7.11}$$

The IPOPT software takes in as input the user-provided routines that compute the cost-function, the gradient of the cost-function and the Jacobian of the constraint equations and returns optimal values for the parameters p_q.

7.2 Application to optimal design of aircraft thermal management system

Figure 7.1: Model of thermal management system.

The heat loads and fuel consumption rates during various modes of the aircraft mission are different. This leads to some interesting design problems. One interesting design problem is to find the optimal fuel flow rates for each mode so that the temperature of the fuel that goes into the nozzles stays close to an optimal temperature. For the purposes of this paper, we consider the design of the TMS as if there were only two modes - *Taxing* and *Flying*. Indeed, these modes are the two most crucial modes of a typical mission as the amount of time spent in other modes like *take-off* and *landing* is extremely short compared to the overall mission time and the contribution of design parameters in these modes to relevant cost-functions is negligible. Therefore we focus on the problem of choosing the optimal fuel flow rates during the *taxing* and *flying* modes and treat the problem as if there were only two modes.

We consider two dynamic variables - the mass (M) and temperature (T) of the fuel remaining in the tank. The temperature of the fuel after it absorbs heat from the EPS and ECS is denoted by T_f. The increase in temperature ($Tf - T$) is related to the fuel flow rate m_{out} through the heat-balance equation

$$m_{out} C_{sp} \left(T_f - T\right) = H_L, \tag{7.12}$$

where H_L is the heat load coming from the ECS/EPS. C_{sp} is the specific heat of the fuel. The fuel that is returned to the fuel-tank is cooled by the air-fuel heat exchanger. The temperature of the

fuel after it is cooled is denoted by T_{in}. The drop in temperature $(T_f - T_{in})$ is assumed to be a fraction of the difference between Tf and the outside air temperature T_{air}. i.e.,

$$T_f - T_{in} = f(Tf - T_{air}), \tag{7.13}$$

where f is referred to as the heat sink efficiency. If m_f is the rate at which fuel is consumed, then the rate at which fuel is returned to the fuel-tank after re-circulation is given as $m_{in} = m_{out} - m_f$. The rate of change of the temperature of the fuel remaining in the tank is derived from the heat-balance equation

$$\begin{aligned} m_{in}C_{sp}T_{in} - m_{out}C_{sp}T &= \frac{d}{dt}\left(MC_{sp}T\right) \\ &= -m_f C_{sp}T + MC_{sp}\dot{T}. \end{aligned} \tag{7.14}$$

Discretizing the above equations, the discrete-time dynamics for the temperature (T) and mass (M) variables can be written as

$$\begin{aligned} M(k+1) &= M(k) - \delta.m_f(k) \\ T(k+1) &= T(k) + \frac{\delta}{M(k)}\left(m_{in}(k)T_{in}(k) - m_{out}(k)T(k) + m_f(k)T(k)\right). \end{aligned} \tag{7.15}$$

Here δ is the size of the discrete time-step and from the above equations we have

$$\begin{aligned} T_{in}(k) &= Tf(k) + f(T_{air} - Tf(k)) \\ \text{and where } Tf(k) &= T(k) + \frac{H_L}{m_{out}C_{sp}}. \end{aligned} \tag{7.16}$$

Note that m_f and m_{out} are considered to be constant within each mode. More precisely

$$m_{out}(k) = \begin{cases} m_{taxi} \text{ if } 0 \le k < \Delta_{taxi} \\ m_{fly} \text{ if } \Delta_{taxi} \le k < \Delta_{taxi} + \Delta_{fly}. \end{cases} \tag{7.17}$$

m_{taxi} and m_{fly} are the parameters that need to be chosen so that we get some desirable thermal behavior. Δ_{taxi} and Δ_{fly} are random variables uniformly distributed within the intervals $[300s, 900s]$ and $[3600s, 4500s]$ respectively. H_L is also considered to be constant within each mode. The cost-function that we are going to use for this problem is a combination of the quality of the fuel temperature going into the combustor (Tf) and the control effort in terms of the fuel flow rates. The cost-function is

$$C = \mathbb{E}\left[\frac{1}{2}\sum_k \left(Tf(k) - T_{set}\right)^2 + \frac{W}{2}\sum_k m_{out}^2(k)\right] \tag{7.18}$$

T_{set} is a set-point temperature at which we desire the fuel-combustor temperature (Tf) to be close to. W is a parameter that decides how much the control effort in terms of the fuel flow rates should be penalized.

As described for the SAA method, we generate a finite number of samples for the taxing and flying times. The samples are Δ_{taxi}^i and Δ_{fly}^i for $i = 1, 2, ..., N$. The sample average cost-function is given as

$$\bar{C} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\sum_k \left(Tf_k^i - T_{set}\right)^2 + \frac{W}{2}\frac{1}{N}\sum_{i=1}^{N}\Delta_{taxi}^i m_{taxi}^2 + \Delta_{fly}^i m_{fly}^2. \tag{7.19}$$

Variable	Value
H_L (Taxing)	$18.44(kW)$
H_L (Flying)	$20(kW)$
m_f (Taxing)	0.84 (kg/s)
m_f (Flying)	1.44 (kg/s)
C_{sp}	0.2 (kJ/kg K)
f	0.1
T_{set}	320K
$M(0)$	9000 kg
$T(0)$	280K

Table 7.1: Fixed parameter values for TMS model.

The gradient of the sample average cost-function with respect to the optimization variables are given as

$$\begin{aligned}
\frac{\partial \bar{C}}{\partial M_k^i} &= 0.0, \\
\frac{\partial \bar{C}}{\partial T_k^i} &= \frac{1}{N}\left(Tf_k^i - T_{set}\right), \\
\frac{\partial \bar{C}}{m_{taxi}} &= \frac{1}{N}\sum_{i=1}^{N}\sum_{k=0}^{\Delta_{taxi}^i}(Tf_k^i - T_{set}).\frac{-H_L}{m_{taxi}^2 C_{sp}} + \frac{W}{N}\sum_{i=1}^{N}\Delta_{taxi}^i m_{taxi}, \\
\frac{\partial \bar{C}}{m_{fly}} &= \frac{1}{N}\sum_{i=1}^{N}\sum_{k=\Delta_{taxi}^i}^{\Delta_{taxi}^i+\Delta_{fly}^i}(Tf_k^i - T_{set}).\frac{-H_L}{m_{fly}^2 C_{sp}} + \frac{W}{N}\sum_{i=1}^{N}\Delta_{fly}^i m_{fly}.
\end{aligned} \tag{7.20}$$

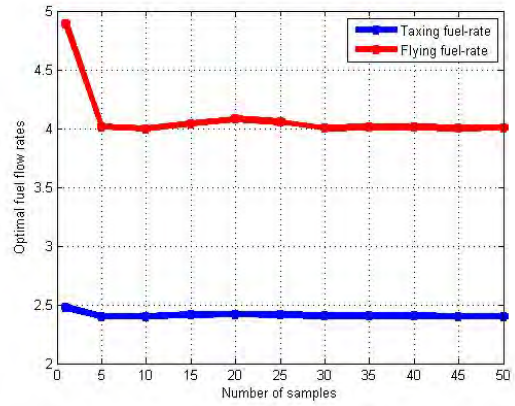
The constraints on the variables M_k^i and T_k^i are derived from the dynamics as described in equation (7.7) of Section 7.1.2. Also the elements of the Jacobian of the constraint equations are computed as described in equation (7.11) of Section 7.1.2.

7.2.1 Results

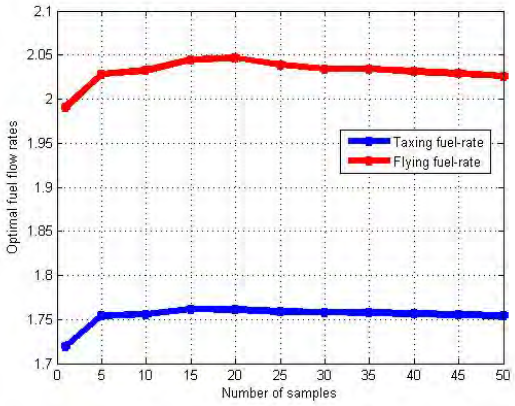
We solved the SAA problem for different number of samples. For 50 samples, the number of constraint equations derived from the system dynamics is around 450,000. For a problem of this size, IPOPT takes about 4 minutes to solve the optimization problem on a laptop with a Intel Core2 Duo CPU P9400 running at 2.40 GHz and 2.95 GB RAM. The values for various fixed parameters in the TMS model are shown in Table 7.1. Figure 9.5 shows some optimization results obtained using IPOPT for different values of W. The plots shows how the optimal fuel flow rates obtained change as the number of samples are increased. As you can see, the optimal values converge very quickly with respect to the number of samples.

For $W = 1.0$, the optimal *taxing* fuel flow-rate (m_{taxi}) is roughly 3 times bigger than the fuel consumption rate (m_f). The value of the cost-function is lowest for this higher flow-rate because the penalty on the control effort is small and for this higher fuel-rate, the resulting fuel temperature (Tf) after it absorbs heat from the ECS/EPS is made lower (and closer to T_{set}). However, if the fuel flow-rate is increased beyond this optimal value, the cost-function would increase because the resulting temperature Tf may go way below the set-point temperature T_{set}. For $W = 250.0$, the optimal value for m_{taxi} is roughly 2 times bigger than m_f. This is because the penalty on the control effort is higher and the fuel-flow rates have to be reduced resulting in higher values for Tf.

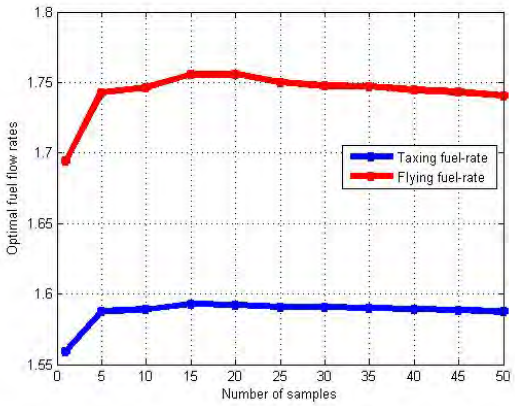
Figure 7.3 shows the variation of the objective function and optimal fuel flow-rates with respect to the weighting parameter W. As described before, the objective function has two components.



(a) W = 1.0



(b) W = 125.0



(c) W = 250.0

Figure 7.2: Results obtained using the SAA method for the optimal design of TMS. These plots show results obtained for different values of W. The plots show how the optimal fuel-flow rates change as the number of samples are increased.

The first component reflects the quality of the fuel temperature going into the combustor and is given as

$$C_1 = \mathbb{E}\left[\frac{1}{2}\sum_k \left(Tf(k) - T_{set}\right)^2\right]. \tag{7.21}$$

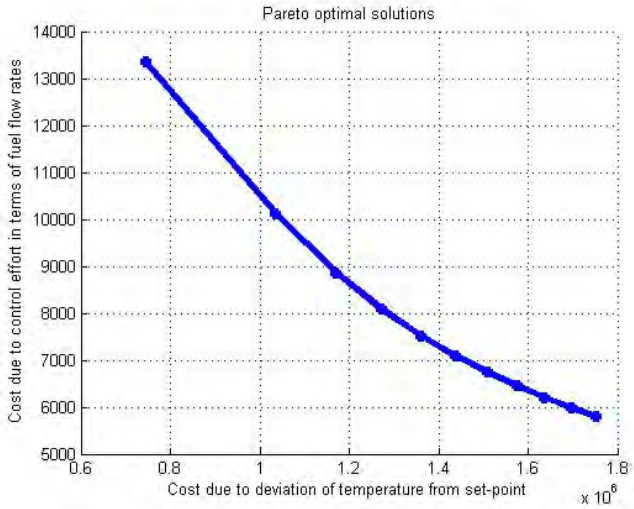
The second component reflects the control effort in terms of the fuel-flow rates and is given as

$$C_2 = \mathbb{E}\left[\frac{1}{2}\sum_k m_{out}^2(k)\right]. \tag{7.22}$$

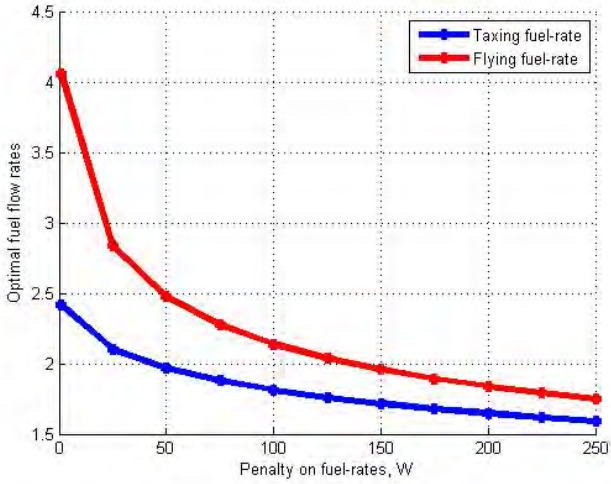
Now for different values of W, we get different Pareto optimal solutions in the following sense. For a given value of W, let p_q^* be the optimal parameters that lead to optimal objective function values of C_1^* and C_2^*. Then for the system to perform such that the objective function $C_1 = C_1^*$, the minimum required value of C_2 is C_2^* and vice versa. Figure 7.3 shows a plot of (C_1^*, C_2^*) for different values of W. Figure 7.3 also shows how the optimal fuel flow rates change with respect to W. It can be clearly seen that the optimal flow rates decrease as W is increased.

7.3 Summary and Future steps

We have described a simple but effective procedure to optimally design hybrid systems whose mode transitions are time-triggered. In particular, we assumed that the transitions times are random variables with known distributions and used stochastic optimization methods to design the hybrid system so that on average, its performance is optimal. We illustrated the method to optimize the fuel flow-rates in different modes of operations of the thermal management system of a prototypical aircraft. In the current setting, we assumed that the parameters to be optimized for are fixed for each mode. It is possible to formulate optimal control problems where the control has some feedback dependence on the state. The feedback dependence on the state can be expressed in terms of parameters and then one could in principle optimize for these parameters.



(a) Pareto optimal curve. The x-axis is for C_1^* as defined in (7.21) and the y-axis is for C_2^* as defined in (7.22).



(b) Variation of optimal fuel flow-rates with W.

Figure 7.3: Results obtained using the SAA method for optimal design of TMS

Chapter 8

Refinement of control and computation platform

In this chapter we review our previous work [45] to model and analyze hardware architectures in a probabilistic setting. We provide a way of capturing stochastic effects arising from variations in performance (e.g. communication delays) as well as failures. In principle, stochastic hybrid systems could be used to model hardware platforms but a more powerful set of tools are available for finite state systems. Our approach is then to operate at different abstraction levels as explained in detail in Chapter 9, where we show that the synthesis of the optimal control laws use vertical contracts coming from the underlying hardware platform. These contracts are the used as specification to design the hardware architecture. The tools presented in this section can be used to check that such contracts are satisfied. This chapter presents one modeling approach for the hardware architecture leading to a Markov Chain model which can then be analyzed using several methods.

8.1 The Stochastic Automata Network Model

A Stochastic Automata Network (SAN) [46] $\mathcal{S} = (\mathcal{A}, E, \Rightarrow)$ comprises a set of stochastic automata $\mathcal{A} = \{A^{(1)}, ..., A^{(n)}\}$, a global set of events E, and a relation $\Rightarrow \subseteq E \times E$. A stochastic automaton is a tuple $A^{(i)}(S^{(i)}, T^{(i)}, L^{(i)}, G^{(i)})$ where $S^{(i)}$ is a set of states, $T^{(i)} \subseteq S^{(i)} \times S^{(i)}$ is a set of transitions, $L^{(i)} : T^{(i)} \to 2^E$ is a labelling function that associates a set of events to each transition. The set of possible system states (not necessarily all reachable) is $S = \times_{i=1}^{n} S^{(i)}$. Let $\Pi(S)$ be the set of all partitions of S and let $\Lambda = \cup_{P \in \Pi(S)}[P \to \mathbb{Q}_+ \cup \mathcal{P} \cup \{\top\}]$ be the set of all functions from partitions to the union of the positive rationals, a set of parameters $\mathcal{P}$, and a special symbol $\top$ which denotes *any* rate. The guard function $G^{(i)} : T^{(i)} \to \Lambda$ associates to each transition a state dependent rate.

The relation among events $\Rightarrow$ (reflexive, antisymmetric and intransitive) imposes restrictions on the set of possible behaviors of the SAN. If $(e_1, e_2) \in \Rightarrow$ we also write $e_1 \Rightarrow e_2$ and we say that e_1 *implies* e_2, or e_2 is *implied* by e_1. To define the semantics of a SAN, we first define its language, i.e. the set of possible computation paths. A computation path is a sequence of states and transition times $\pi = (s_0, \tau_0, s_1, \tau_1, ..) \in (S \times \mathbb{R}_+)^\omega$. Such path is valid if it satisfies the following set of conditions expressed in terms of the state transitions $t_i = (s_i, s_{i+1})$: 1) $t_i^{(k)} \in T^{(k)}$ where we indicate with $t_i^{(k)}$ the projection of t_i on the states of the k-th automaton, i.e. $t_i^{(k)} = (s_i^{(k)}, s_{i+1}^{(k)})$; 2) $G^{(k)}(t_i^{(k)})([s_i]) \neq 0$, where $[s_i]$ is the partition containing s_i; 3) Either $L^{(i)}(t_i^{(k)}) = \emptyset$ or, $\forall\ e \in L^{(i)}(t_i^{(k)})$, if e is implied by some e' then there must be a transition $t_i^{(j)}$ for some automaton $A^{(j)}$ such that $e' \in L^{(j)}(t_i^{(j)})$. Time t_i is the time spent in state s_i and depends on the transition rate at s_i. The transition rate is not straightforward to define. Consider two events in a relation $e_1 \Rightarrow e_2$ and the two

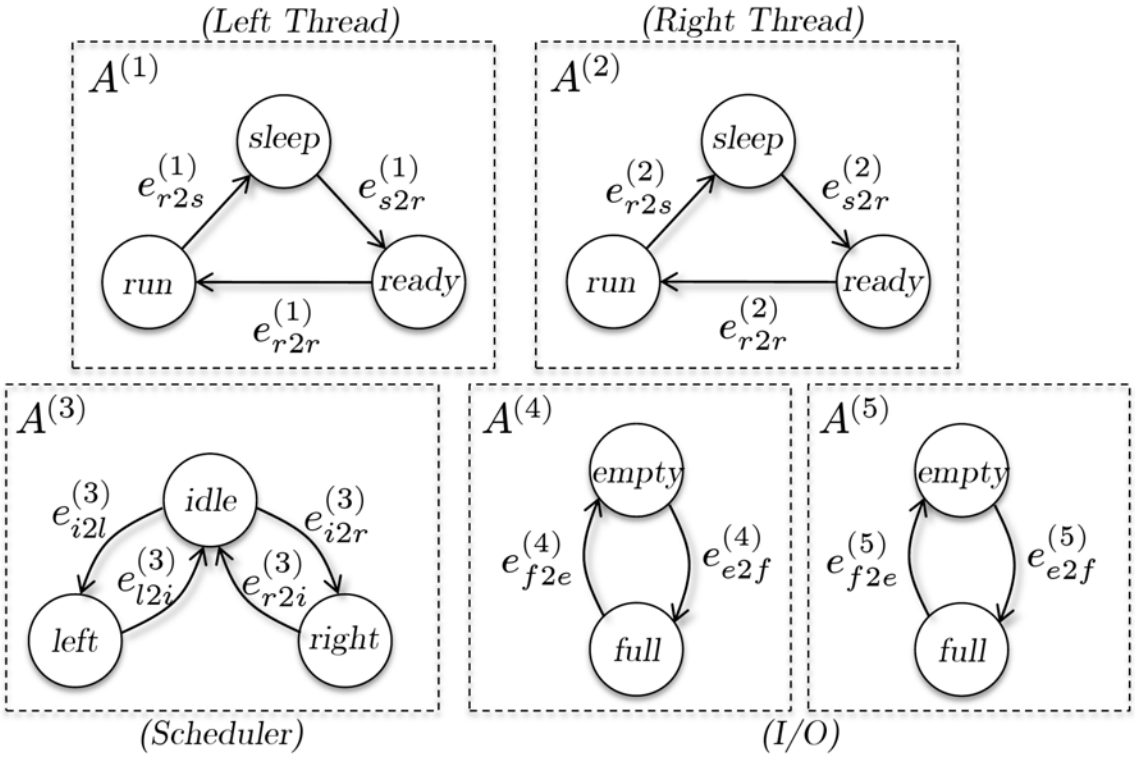


Figure 8.1: Example of model of a processing element with two threads a scheduler and I/O buffers.

respective transitions $t_1(s_i^{(k)}, s_{i+1}^{(k)}) \in T^{(k)}$ and $t_2(s_i^{(j)}, s_{i+1}^{(j)}) \in T^{(j)}$. The rates of this transitions are $G^{(k)}(t_1)([s_i])$ and $G^{(j)}(t_2)([s_i])$, respectively, which might be different. During reachability analysis, the rate of the implied transition will be constrained to be equal to the rate of t_1. For illustration purposes and to keep the exposition simple, we will use *binary* guard functions. A binary guard function maps a transition to a subset $\Lambda' \subset \Lambda$, where Λ' only contains mappings whose domains are the binary partitions of the state space. Moreover, one element of the partition must map to zero. We refer to binary guard functions as to *guard conditions*.

The language of a SAN is the set of all valid computation paths. It is easy to see that each computation path is the realization of a Markov Process. In fact, a SAN can be described by its underlying Continuous Time Markov Chain (CTMC). For a characterization of the probability space defined by a CTMC, please refer to [12]. The type of SAN presented in this section provides the basic elements to capture complex synchronization patterns among the transitions of the automata of the SAN.

8.2 Examples of architectural components

A model of a processing elements is shown in Figure 8.1. The model comprises two threads, a scheduler, and two I/O buffers (one for transmitting, and one for receiving data). The thread model is an automaton with three states: in the *sleep* state the thread is inactive; in the *ready* state the thread is ready to be executed, but needs to wait to get ownership of the shared computational resource; in the *run* state the thread is executed on the processing element. This model is an abstraction of the thread model defined by the AADL language (one of the few behavioral aspects

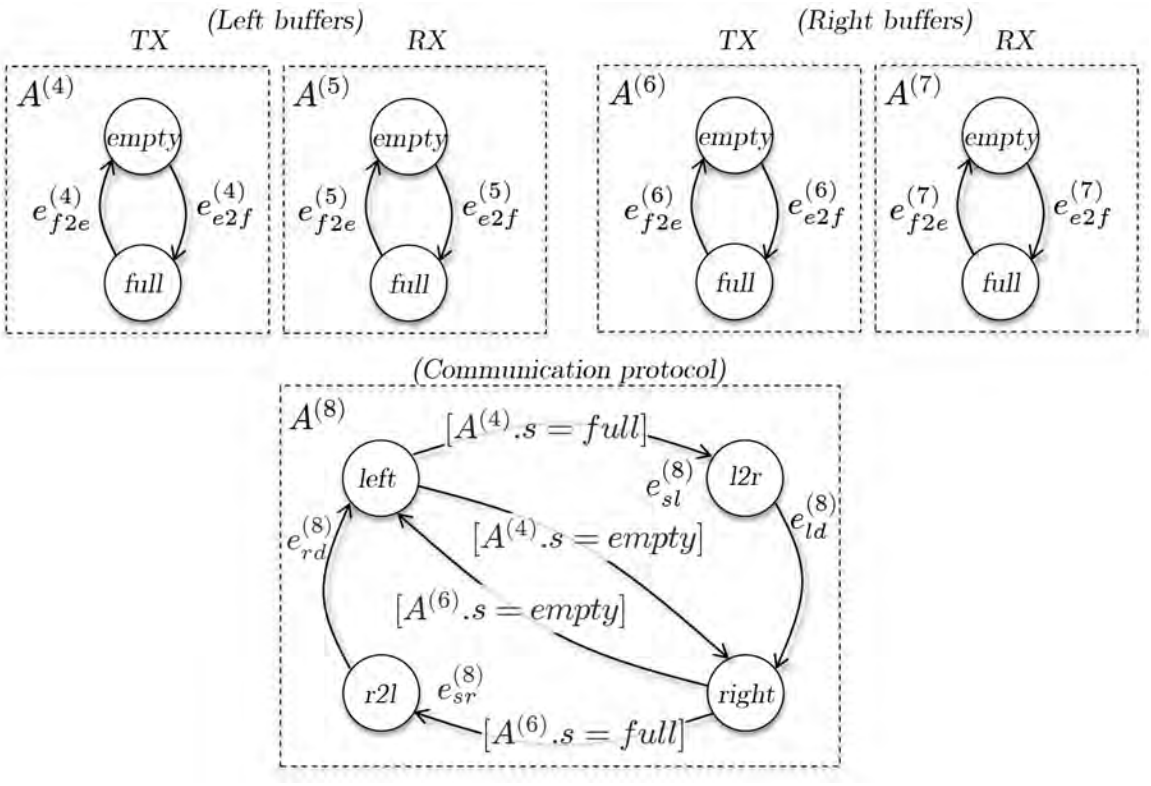


Figure 8.2: Example of model of I/O buffers and a communication protocol.

included in the standard). The scheduler implements a first-come-first-served policy. We have labelled each transition with one event but we have not shown guard conditions.

The thread activation policy determines when the transition from the *sleep* state to the *ready* state occurs. For example, a thread in the AADL language is associated with a dispatch protocol property. A thread can be dispatched periodically, aperiodically, sporadically, or it can be dispatched only once until completion. The transition from the *ready* state to the *run* state is driven by the scheduler. The scheduler decides which thread takes ownership of the processor. The scheduler starts from the *idle* state and it can transition to the $left$ or $right$ states to grant the exclusive use of the processing element to the left or right thread, respectively. Consider transition $(idle, right)$. This transition is only taken when the right thread is ready to run, which correspond to the set of system states $S_{rr} = \{s \in S | s^{(2)} = ready\}$. Thus $G^{(3)}(idle, right)(\neg S_{rr}) = 0$ while $G^{(3)}(idle, right)(S_{rr})$ corresponds to the overhead associated with loading the context of the new thread to be executed. When the transition is taken, the thread must move to the *run* state, thus $e^{(3)}_{i2r} \Rightarrow e^{(2)}_{r2r}$. The thread can now be executed. When the execution is over, the thread transitions back to the *sleep* state and releases the resource, i.e. $e^{(2)}_{r22} \Rightarrow e^{(3)}_{r2i}$. Guard conditions and synchronizations are similarly defined for the left thread. The I/O buffers are one place buffers directly used by the application software while in the *run* state. In this example we showed a first-come-first-served scheduler, but other schedulers can be modeled. Also notice that performance metrics are captured by exponentially distributed transitions. This is not realistic in general and there are techniques to deal with this problem such as the use of phase-type distributions [43], or the solution of the underlying Markov Regenerative Process [30].

Figure 8.2 shows the model of a communication protocol that allows to transfer data between the

λ	System	$P(sleep)$	$P(ready)$	$P(run)$
	$sys2$	0.275	0.058	0.666
8000	$sys4$	0.380	0.038	0.581
	$sys4f$	0.371	0.038	0.591
	$sys2$	0.378	0.040	0.581
4000	$sys4$	0.453	0.025	0.522
	$sys4f$	0.439	0.025	0.535
	$sys2$	0.459	0.026	0.515
2000	$sys4$	0.505	0.016	0.479
	$sys4f$	0.489	0.016	0.495

Table 8.1: Probabilities of being in the sleep, ready or run state at $t = 1ms$ for thread th_2.

I/O buffers of two threads. The protocol model is not so different from the scheduler of Figure 8.1. In fact, it manages access to a communication medium (the shared resource) from multiple sources. The scheduling policy implemented by the protocol is token-ring. The initial state is $left$ meaning that the left I/O TX buffer is checked first. If it is empty, then the protocol passes to check the right buffer. If the left buffer is full, then it is served by moving the message to the right receiving buffer. This is achieved by two synchronizations as follows: $e_{sl}^{(8)} \Rightarrow e_{f2e}^{(4)}$ and $e_{ld}(8) \Rightarrow e_{e2f}^{(7)}$. While the model seems intuitive, the difficulty lies in the potentially large set of implementation options associated with this simple transfer. To mention a couple, we have assumed that the routing of messages is statically defined. This allows to solve the transfer of messages among queues using synchronizations only. As a matter of fact, we are not even defining messages. The definition of message types in the architecture is only used to determine the average transfer delay associated with the transition $(l2r, right)$. Further, we have assumed an overwriting policy for the buffers, i.e. the protocol does not check whether the RX buffer is full before executing transition $(l2r, right)$.

8.3 Example of architectural analysis

We consider a distributed architecture composed of processors running a single thread and communicating over a token ring bus. This architecture is built using the templates presented in Section 8.2. Each thread th_i transitions from the *sleep* state to the *ready* state when the transmission buffer TX_i is empty. When the thread is scheduled to run, it first reads from buffer RX_i, and then writes to TX_i. The token ring bus serves the TX buffers and broadcasts their content to all RX buffers in the system. We consider transition rates of 10^5, 10^4 and 10^3 for transitions $(sleep, ready)$, $(ready, run)$ and $(run, sleep)$ respectively. We also consider a rate of 8000 for the protocol to pass the token among users, while we leave the data transmission rate λ as a parameter (to mimic the effect of different packet sizes). We consider three architectures: $sys2$ with two processors (182 reachable states), $sys4$ with four processors (24708 reachable states) and $sys4f$ with 4 unreliable processors (2118680 reachable states). Unreliable processors can fail with rate 0.0003, and recover from failure with rate 0.3. The results of the analysis are shown in Table 8.1 where we report the probability of being in the sleep, ready or run state for thread th_2 at time $t = 1ms$.

The results show two obvious trends. When the number of processors increases, the token rotation time increases and the time a task spends in the sleep state also increases. If the transmission time increases, the time spent in the sleep state also increases. Interestingly, the time spent in the run state is higher for $sys4f$ than for $sys4$. This is because thread th_2 can leverage the time when other processors are silent because of a failure.

These results can be used to determine the execution rates of a tread mapped on a process or the maximum transmission rate of data from a sensor. The result can then be used to check whether the architecture is capable of supporting a given control function.

Chapter 9

Towards optimal design of the control, computation and communication platforms

In this chapter we describe our methodology for correct-by-construction design of the control, computation and communication platforms. Some previous work in this area will be reviewed that are mainly concerned with the optimization of hardware platform for metrics such as cost and extensibility. The novelty of our approach is twofold: 1) we will define metrics that are related to *vulnerability* in a general sense, namely the probability that the system will catastrophically fail to deliver the desired level of performance and we will optimize the control, computation and communication platform to minimize vulnerability; 2) we will start the design exploration activity at a much higher level, namely the design of distributed control architectures. The problem of exploring the functional architecture of the control algorithm (from centralized to distributed) is still an open problem. In fact, there is no systematic way of approaching such problem. A typical approach consist in first designing a distributed control algorithm that solves a given control problem and then analyzing its properties to check whether control requirements are met. We will provide application examples of our design flow.

9.1 Introduction

A complex dynamical system is the composition of blocks that represent mechanical and electrical components. These blocks have their dynamics described by a set of differential equations $\dot{x} = f(x, u, u_c)$ where in general x is the state vector associated with that block, u is a vector of input coming from other blocks and u_c is a set of control inputs that can be forced by a controller. We call this interconnection of blocks the *physical architecture* (Figure 9.1).

The *cyber architecture* is decomposed into two levels: the control architecture and the embedded execution platform architecture. The control architecture is the result of the composition of several agents that implement control functions. Such agents cooperate to drive the dynamic evolution of the physical agents towards a desired behavior. Controllers generate signals $u_{c,i}$ for each physical agent and sense (either directly or indirectly) their state x_i. The control functions are implemented by processing elements in the embedded platform. Figure 9.1 a typical example where all control agents are mapped on a triple redundancy computing platform. A communication network is also present to implement the communication among sensors, actuators and control blocks.

The correct operation of the system depends on several factors. We are concerned with the probability that components may fail rather than with the correctness of the control algorithms.

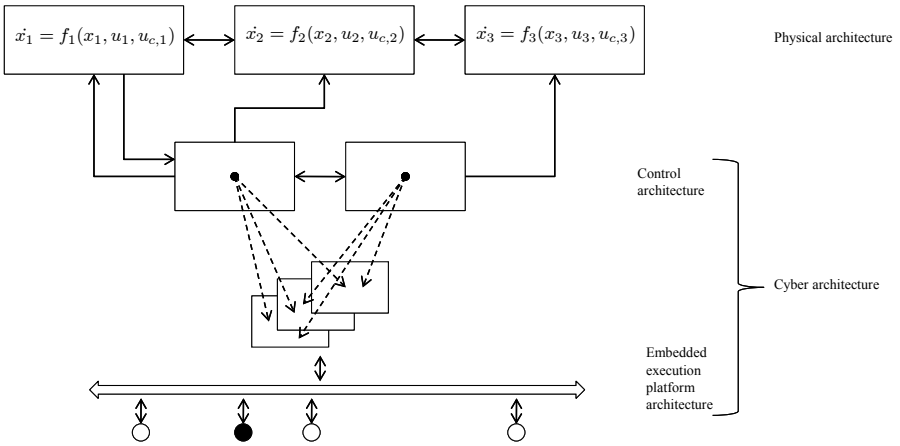


Figure 9.1: The levels of a cyber-physical architecture.

In fact, we assume that controllers can be synthesized and we will also show a methodology for correct-by-construction control design later in this chapter. Failure is associated with hardware components such as the mechanical equipment in the physical architecture, the processing boards, communication links, sensors and actuators. We consider several failure models:

- For hardware failures we may have two different models. If the failure is permanent, then the component is no longer available after the failure occurs. This is the case where there is no redundancy in the system which means that there is no other component that can be used as a replacement. If redundancy is built in the system, then a component may fail and recover after a certain amount of time. In this case there is a limit on the maximum number of consecutive failures. The failure probabilities can be directly obtained by looking at the mean time before failure of a component.
- We consider damages as particular types of failures. Damages are permanent failures but their occurrence is determined by external events and the use of a probabilistic model may or may not be the best way of representing them.
- Performance failures are deviations from nominal performance values. Performance failures can be characterized using a probabilistic model, meaning a probability distribution around the nominal value. A typical example is network delay. Chapter 8 discusses some examples of modeling the performance variations.

We will detail these failure models in the following sections and we will then abstract them at up to the control architecture level so that controller analysis and synthesis can be performed without the additional complexity induced by the embedded platform. This abstraction will allow us to trade-off centralized versus decentralized control strategies with the objective of minimizing vulnerability while providing acceptable performance.

9.2 Abstraction of the embedded platform

The complexity associated with a joint exploration of the control architecture and the embedded platform architecture makes the problem intractable. The decomposition of the two problems relies

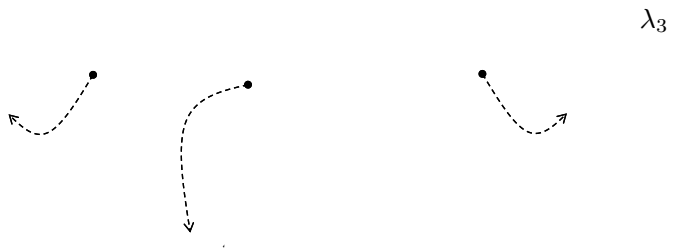


Figure 9.2: Example of a physical architecture with a three-agent controller. Each controller is associated with a state machine representing the failure mode (`OP` means that the controller is operational while `FAIL` means that the controller is faulty).

on the abstraction of the embedded platform metrics at the control level. The abstraction is created as a result of matching the semantics of the language used for the description of the control algorithm with the fault model of the platform.

Both the physical architecture and the control algorithm are described using the continuous time model. Each agent is defined in terms of a set of differential equations. Agents communicate over shared signals (i.e. continuous variables that are shared among the blocks). The semantics of the composition of agents is defined by the flat set of differential equations that can be obtained by imposing equality constraints between the outputs and inputs whenever they are connected.

We first abstract the failure probabilities of processing and communication elements. Consider an agent that is part of the control architecture. The agent will be ultimately executed by a processing element that may fail. When the processing element fails, the agent is not available anymore. We assume that the failure is silent, meaning that the failed processor does not produce erroneous data, but rather stops transmitting. In this case, another control agent that expects data from the faulty agent realizes (perhaps using timeouts) that input updates are not available and assumes for them some nominal values. Another strategy for the control agent is to use the last received input for future computations. These two failure models can both be adopted with their advantages and disadvantages. Notice, for instance, that resorting to a nominal value may represent a step change at the input of a controller.

Figure 9.2 shows a system where the control architecture has three control agents C_1, C_2 and C_3. They can sense the state of the agents in the physical architecture and can send control commands. We have abstracted the interconnection among the control agents into a single block that, as will be detailed later, is also part of the design of the decentralized control algorithm. Each controller is associated with a state machine that captures the failure modes of that controller. In the `OP` mode, the control agent is operating normally as intended by design. In the `FAIL` mode, the agent is faulty and its outputs are either kept constant to the value right before the failure or they are set to a nominal value. We capture permanent faults, as in the case of C_2 and C_3, and transient faults, as in the case of C_1. Using multiple `OP` states and `FAIL` states allows to model systems than can fail and recover from failure a bounded number of times.

The stochastic automaton that capture the failure modes of the different control agents can be composed to yield a stochastic automaton that captures the failure modes of the composed system. The model described in [46] is an effective way of capturing not only independent faults, but also

faults that are correlated. For example, suppose that both C_2 and C_3 are mapped on the same computing resource. Then, their failures happen simultaneously, meaning that when C_2 transitions to the `FAIL` state, so does C_3 and vise-versa. Each failure state at the system level correspond to a different controller applied to the physical architecture because each faulty control agent changes the dynamics of its outputs to be constant at a nominal value. Thus, the performance of the decentralized control algorithm also changes depending on the failure mode.

A similar model can be used to abstract failures associated with the communication network. Each link in the decentralized control architecture can be associated with a failure rate. The failure rate can be though of as a packet corruption or a packet drop (transient failures) or as a permanent damage to the network link. If a link fails, each control agent that uses that link as input assumes either a nominal value or the last known good value.

The abstraction of the performance failures turns out to be more complicated. Modeling delays in the communication network (for example) and including this effect in the dynamics of the system leads to differential equations which are difficult to solve. In practice, the implementation of decentralized control algorithms is based on timeouts. If an input is not received within a given time window, then it is assumed that the packet has been dropped and therefore an excessive delay can be modeled as a packet drop (i.e. a transient failure). The probability of this event to occur can be computed for a given communication delay distribution.

9.3 Vulnerability metric and analysis method

We will define vulnerability in terms of the probability that the system fails catastrophically. Events that can cause such type of failures are application dependent and need to be identified by designers. For air vehicles, the inability to fly safely is of major concern. Thus, for these type of systems, any event that causes the vehicle to crash is catastrophic. We realize that this approach to the definition of vulnerability may seem subjective. However, we believe that this initial event classification which is application dependent is unavoidable.

We distinguish between a catastrophic event and its effect. The effect is the manifestation of the occurrence of the event which leads to the undesired result of a crash. The event is defined as the transition into a state from which the system cannot recover and that leads to a crash. The state of the system is in general hybrid as defined in Chapter 2. Let $\mathcal{S} = \cup_{q \in Q}\{q\} \times \mathbb{R}^{d(q)}$ be the hybrid state associated with a system. A catastrophic event can then be defined as the transition of the system into a subset of the state space $\mathbb{B} \subseteq \mathcal{S}$ called bad set. We are interested in a bounded time window Δ (also called horizon) which is typically identified by the length of a mission. Therefore, we define vulnerability as follows:

Definition 21 (Bounded vulnerability)**.** *Given a stochastic hybrid system $\mathcal{H}$ with hybrid state space $\mathcal{S}$ and a bad set $\mathcal{B} \subseteq \mathcal{S}$, the bounded vulnerability of $\mathcal{H}$ is $P(\{\tilde{s}(t)\}_{t \leq \Delta} \in \mathcal{B})$, i.e. the probability that the stochastic process $\tilde{s}(t)$ generated by $\mathcal{H}$ enters the bad set before time Δ.*

This definition provides a way of computing vulnerability of a system. Once the bad set has been defined, vulnerability entails solving the reachability problem for the stochastic hybrid system which is in general hard (as shown in Chapter 2). The analysis can be simplified in the case of continuous controllers where the discrete transitions of the hybrid system are only determined by failure rates. In this case, transitions do not depend on the continuous states and more efficient algorithms can be developed to compute vulnerability.

In the propose approach, we start from a model of the physical and control architecture as a set of continuous systems. Then, we associate failure models to each components. The combination of these two models leads to a hybrid dynamical system where mode changes are probabilistic. Moreover, when an agent or a communication link fails, the dynamics of the system is simplified because some of the variables are kept constant.

The problem that we will address is the synthesis of the distributed control architecture taking into account physical and the embedded architecture. The synthesis problem asks for the decentralized control architecture representing an optimal trade-off between vulnerability and cost.

9.4 Control/Platform Architecture exploration for thermal management system

In this section, we apply a synthesis methodology to the thermal management system (TMS) described in the previous chapters. The objective is to develop tools that are able to derive automatically a system which satisfies some properties which, therefore, do not need to be verified. We also aim at exploring the design space by selecting promising systems which satisfy those properties while being optimal with respect to cost. Using the TMS as an example, we describe a two-step process to perform the design exploration. The first step is to design the control architecture that is implemented on the embedded platform to make the physical system perform as desired. In this step, all control architectures are enumerated and the control algorithm that can be implemented using each control architecture is optimized using tools from optimal control theory. The second step is to study the effect of the platform features (like processing element failure rates) on the performance of the controlled system.

The control architecture and the embedded platform architecture can either be distributed or centralized in nature. The degree of decentralization for both the control architecture and the platform architecture can be varied and should be considered a part of the design process. Note that the choice of the control architecture affects the design of the communication architecture and other platform design parameters. The control architecture provides the constraints for the communication synthesis. There is no unique platform architecture on which to implement a given control architecture. Among the various platform architectures on which it is possible to implement a control architecture, we need to choose the one that minimizes some desired metric.

There may be various costs associated with the choice of the control architecture and platform. Some of them are :*computational cost, communication cost, vulnerability* and *performance optimality*. The table below gives a rough summary of the effects of various combinations of control architectures and platforms architectures on these various objectives.

Control \ Platform	Decentralized	Centralized
Decentralized	Low computation cost Low communication cost Performance not optimal Vulnerable	Low computation cost High communication cost Performance not optimal More vulnerable
Centralized	High computation cost High communication cost Optimal performance Least vulnerable	High computation cost High communication cost Optimal performance Vulnerable

9.4.1 Control Architecture Design Problem

First we discuss the design of the control architecture. The control architecture is the result of the composition of the controller agents that implement the control functions. The controller agents cooperate to drive the dynamic evolution of the physical system as desired. A key element in the design of the control architecture is to decide which states of the controlled system should each controller agent have feedback dependence upon. For example, consider the following controlled dynamical system:

$$\begin{aligned}
\dot{x}_1 &= f_1(x_1, x_2, ..., x_N, u_1) \\
\dot{x}_2 &= f_2(x_1, x_2, ..., x_N, u_2) \\
&... \\
&... \\
\dot{x}_N &= f_N(x_1, x_2, ..., x_N, u_N)
\end{aligned} \tag{9.1}$$

Each $x_i \in \mathbb{R}^{m_i}$ represents the state of the i-th subsystem. Each subsystem may correspond to a physical component like a pump or heat-exchanger and the components of the vector x_i correspond to internal variables for that physical component. The dynamics of the state for each subsystem possibly depends on the states of every other subsystem. But typically, the dynamic dependence is sparse. In other words, the Jacobian $\frac{\partial f}{\partial x_i}$ is sparse. The inputs u_i directly controls the dynamics of only the i-th subsystem. But the controls u_i could have some feedback dependence on the states of the other subsystems. The key step in the design of the control architecture is deciding which subsystems the control u_i should have feedback dependence on. For example, if the control algorithm is completely decentralized, u_i would be such that

$$u_i = K_i(x_i). \tag{9.2}$$

i.e., the control u_i has feedback dependence only on the state of the local sub-system. For a completely centralized algorithm u_i would have feedback dependence on the states of all sub-systems. Formally that means

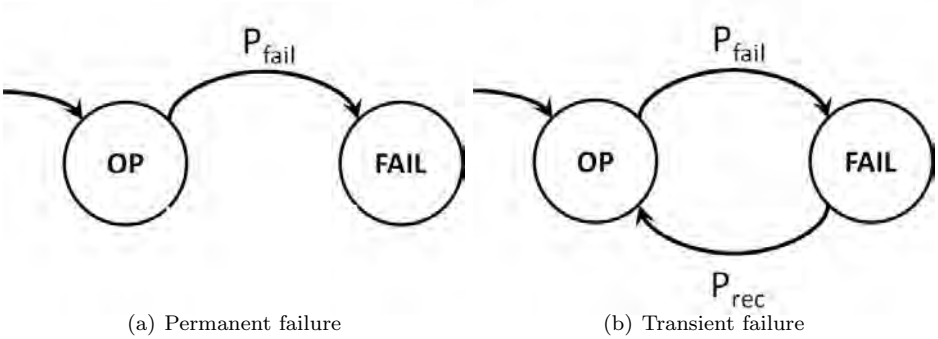
$$u_i = K_i(x_1, x_2, ..., x_N). \tag{9.3}$$

Note that once the feedback dependence is decided, one still needs to design the feedback laws so as to obtain some desirable performance. In other words, one needs to 'shape' the functions K_i so that the system performs in a desired manner. This can be done using tools from optimal control theory as described in the following example . The feedback dependence can be represented using a graph. Consider the directed graph $U = (V_U, A_U)$ where each element in the set of nodes V_U represents a sub-system. If the arc (i, j) is an element of the set A_U, this means that the state of sub-system i influences the control action in sub-system j. In other words, the controller for sub-system j has some feedback dependence on the state of sub-system i. For a fully decentralized control architecture, the graph U would be an edgeless graph. For a fully centralized control architecture, the graph U would be a strongly connected graph. The control architecture design involves the joint optimization of the graph U and the corresponding functions K_i that such that some performance metric is minimized.

9.4.2 vulnerability Optimizaiton Problem

As discussed in Section 9.2, the control functions are implemented on processing elements in the embedded platform. Permanent faults and transient faults can be modeled using the stochastic automata shown in Figure 9.3. If the physical system can be modeled as a stochastic hybrid system, the failure models for the processing elements can be composed with the stochastic hybrid model for the physical system to give a stochastic hybrid model for the entire controlled system. This controlled system can be analyzed using tools like reachability analysis or statistical model checking algorithms (as described in the previous chapters) to compute the vulnerability of the system.

Assume we associated a failure models to each component in the system. Notice that the failure model can capture actual mechanical and electronic failures, or possible battle damages. A more reliable component could be one where the probability of failure is lower or that can fail and recover multiple-times (e.g. redundant hardware). The optimization problem is to minimize the cost of the system subject to vulnerability constraints. The cost function depends on the following parameters:



(a) Permanent failure (b) Transient failure

Figure 9.3: Stochastic automata to model failures of processing element.

- Failure rate: We assume that the cost increases exponentially with the inverse of the failure rate.
- Number of recoveries. We assume that the cost is linear in the number of times an agent can recover from a failure.

9.4.3 Design exploration of TMS

We apply the design methodology described above to the TMS described in previous chapters. The TMS can be thought of as a composition of four subsystems. The four subsystems are *fuel tank, fuel pump, fuel-oil heat exchanger* and *fuel-air heat exchanger*. Figure 9.4 shows a schematic of these four subsystems together with the internal states associated with each subsystem. The blue solid arrows indicate physical connections between the subsystems and the red dashed arrows indicate links in the control architecture. The *fuel pump* and *fuel-air heat exchanger* have local controllers associated with them. The controller for the *fuel pump* can change the fuel flow-rate (m_{out}) depending on the state of system. The controller for the *fuel-air heat exchanger* changes the heat sink efficiency (f) based on the state of the system.

The dynamics for the states in the controlled TMS are described by

Physical states of TMS:

$$\begin{aligned} \dot{M} &= -m_f \\ \dot{T} &= \frac{1}{M}\left(m_{in}T_{in} - m_{out}T + m_f T\right) \\ \text{where } T_{in} &= Tf + f\left(T_{air} - Tf\right) \\ \text{and where } Tf &= \frac{HL}{m_{out}C_{sp}} + T \end{aligned} \tag{9.4}$$

Fuel-pump controller:

$$\dot{m}_{out} = G_1(Tf - T_{set}) - G_2(f)$$

Fuel-air HEX controller:

$$\begin{aligned} \dot{f} &= 0.5cos^2(\theta)\left(L_1(T - \bar{T}) - L_2(m_{out} - m_f)\right) \\ \dot{\theta} &= cos(\theta)\left(L_1(T - \bar{T}) - L_2(m_{out} - m_f)\right) \end{aligned}$$

The dynamics of the auxiliary variable θ is chosen so that $\dot{f} = 0.5cos(\theta)\dot{\theta} = \frac{d}{dt}(0.5sin(\theta))$, and therefore $f(t) = 0.5 + 0.5sin(\theta(t))$, if $\theta(0)$ is chosen to be $sin^{-1}\left(\frac{f(0)-0.5}{0.5}\right)$. Thus $f(t)$ is guaranteed

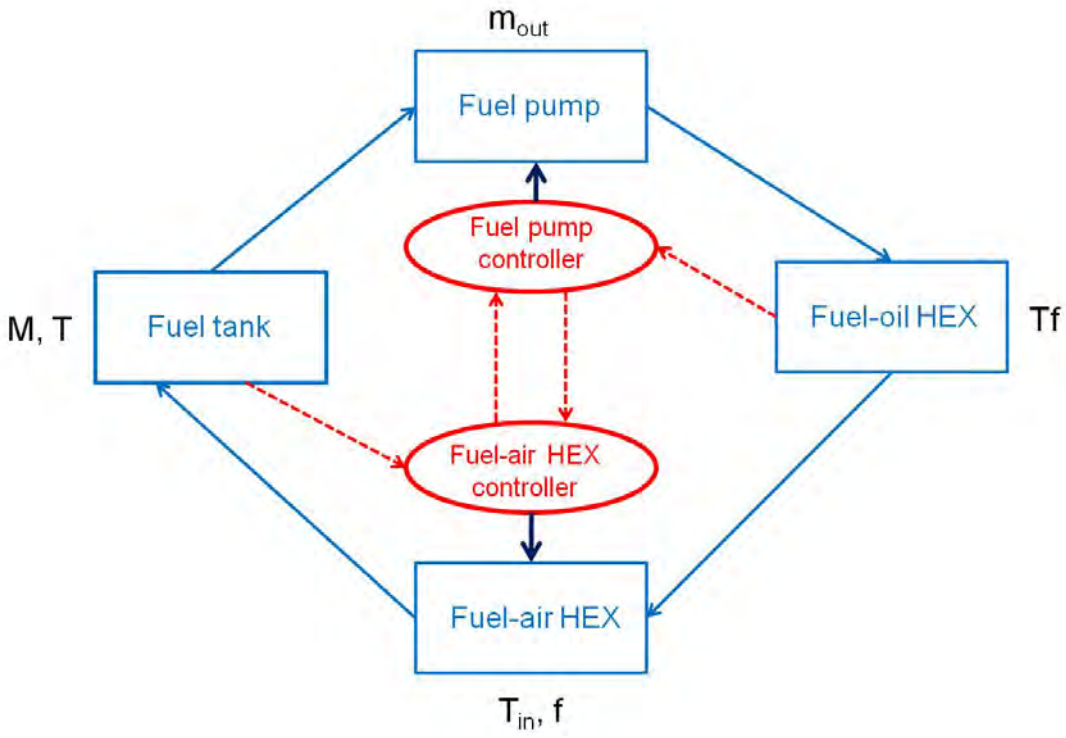


Figure 9.4: TMS as a composition of four sub-systems. The blue solid arrows indicate physical connections between the subsystems and the red dashed arrows indicate links in the control architecture.

to always remain bounded (i.e. $0 \leq f(t) \leq 1$).The control parameters $G_1, G_2, L_1, L_2 \geq 0.0$ are feedback gains that need to be optimized for. T_{set} is a set-point temperature at which we desire the fuel-combustor temperature (Tf) to be close to. $\bar{T}$ is a set-point temperature at which we desire to keep the fuel-tank temperature. The feedback dependence on Tf is such that the fuel flow-rate (m_{out}) increases if Tf is above T_{set} and vice versa. Similarly, the feedback dependence on T is such that the heat sink efficiency (f) increases if T is above $\bar{T}$ and vice versa.

For control architecture exploration, we consider all possible combinations of feedback dependencies. For a given control architecture, we optimize for the control parameters so that a performance metric is minimized. For the control architecture exploration, one of the feedback dependencies can be made to be absent by enforcing a constraint on the corresponding feedback gain. For example, by enforcing the constraint $0 \leq L_1 \leq 0$ in the optimization procedure, we make the dynamics of the *fuel-air HEX* to be independent of the fuel-tank temperature.

For a fixed control arhitecture, we optimize for the control parameters G_1, G_2, L_1 and L_2. We use a similar optimization procedure as described in Chapter 7 to find the optimal control parameters. We use IPOPT ([52]) to perform the optimization. The cost-function used is a weighted sum of the deviations of the fuel temperatures and the magnitude of the fuel-flow rates. More precisely, the cost-function is

$$C = \frac{1}{2}\sum_k \left(Tf^k - T_{set}\right)^2 + \frac{1}{2}\sum_k \left(T^k - \bar{T}\right)^2 + \frac{W}{2}\sum_k \left(m_{out}^k\right)^2 \tag{9.5}$$

The variables that need to be optimized for include both the state variables $M^k, T^k, m_{out}^k, f^k, \theta^k$ and the control parameters G_1, G_2, L_1 and L_2. In our notation, T^k is the state of the system at time k. Following the procedure described in Chapter 7, we discretize the differential equations and derive the constraints imposed by the system dynamics. The constraints can be written as:

$$\begin{aligned}
g_1^k &= M^{k+1} - \left(M^k - \delta.m_f(k)\right) = 0 \\
g_2^k &= T^{k+1} - \left(T^k + \frac{\delta}{M^k}\left(m_{in}^k T_{in}^k - m_{out}^k T^k + m_f^k T^k\right)\right) = 0 \\
g_3^k &= m_{out}^{k+1} - \left(m_{out}^k + \delta\left(G_1(Tf^k - T_{set}) - G_2(f^k)\right)\right) = 0 \\
g_4^k &= f^{k+1} - \left(f^k + \delta 0.5 cos^2(\theta^k)\left(L_1(T^k - \bar{T}) - L_2(m_{out}^k - m_f^k)\right)\right) = 0 \\
g_5^k &= \theta^{k+1} - \left(\theta^k + \delta cos(\theta^k)\left(L_1(T^k - \bar{T}) - L_2(m_{out}^k - m_f^k)\right)\right) = 0
\end{aligned} \tag{9.6}$$

Here δ is the size of the discrete time-step and from the above equations we have

$$\begin{aligned}
T_{in}^k &= Tf^k + f(T_{air} - Tf^k) \\
\text{and where } Tf^k &= T^k + \frac{H_L}{m_{out}^k C_{sp}}.
\end{aligned} \tag{9.7}$$

Note that the rate of fuel consumption (m_f) is constant within each mode. More precisely

$$m_f^k = \begin{cases} m_f^{taxi} \text{ if } 0 \leq k < \Delta_{taxi} \\ m_f^{fly} \text{ if } \Delta_{taxi} \leq k < \Delta_{taxi} + \Delta_{fly}. \end{cases} \tag{9.8}$$

Since the cost-function does not depend explicitly on the control parameters, we have

$$\frac{\partial C}{\partial G_1} = \frac{\partial C}{\partial G_2} = \frac{\partial C}{\partial L_1} = \frac{\partial C}{\partial L_2} = 0.0. \tag{9.9}$$

The gradient of the cost-function with respect to the temporal states is given as

$$\begin{aligned}
\frac{\partial C}{\partial M^k} &= 0.0, \\
\frac{\partial C}{\partial T^k} &= \left(Tf^k - T_{set}\right) + \left(T^k - \bar{T}\right), \\
\frac{\partial C}{\partial m_{out}^k} &= \left(Tf^k - T_{set}\right) \frac{-H_L}{(m_{out}^k)^2 C_{sp}}, \\
\frac{\partial C}{\partial f^k} &= 0.0, \\
\frac{\partial C}{\partial \theta^k} &= 0.0.
\end{aligned} \tag{9.10}$$

Similarly the Jacobian of the constraint equations are computed. Some representative elements of the Jacobian matrix are given below.

$$\begin{aligned}
\frac{\partial g_2^k}{\partial m_{out}^k} &= \frac{-\delta}{M^k}\left((T_{in}^k - T^k) + m_{in}^k(1-f^k)\frac{-H_L}{(m_{out}^k)^2 C_{sp}}\right), \\
\frac{\partial g_3^k}{\partial G_1} &= -\delta(Tf^k - T_{set}), \\
\frac{\partial g_4^k}{\partial L_2} &= -\delta 0.5 cos^2(\theta^k) \times -(m_{out}^k - mf^k).
\end{aligned} \tag{9.11}$$

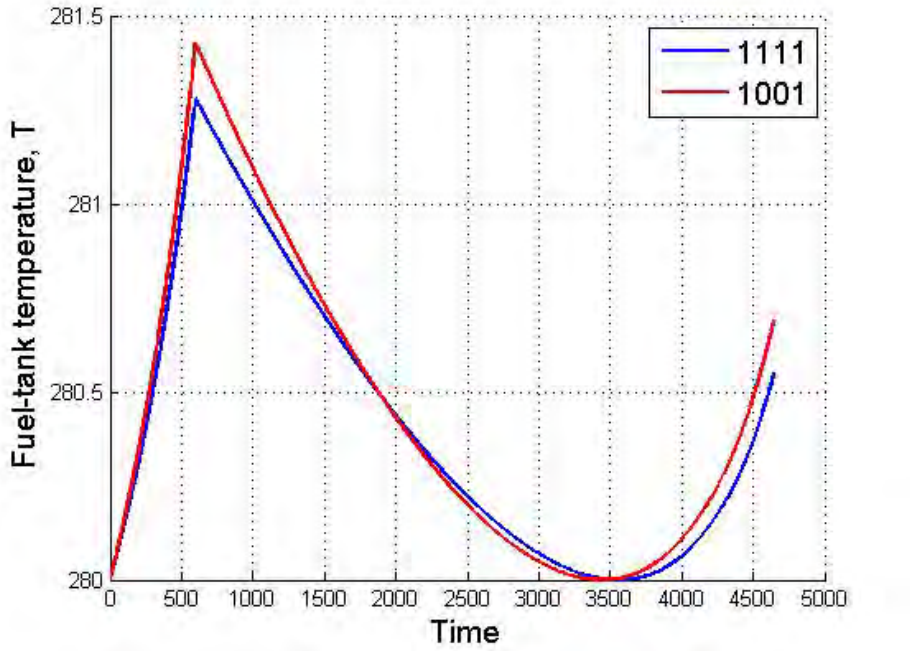
The IPOPT software takes in as input the user-provided routines that compute the cost-function, the gradient of the cost-function and the Jacobian of the constraint equations and returns optimal values for the control parameters G_1, G_2, L_1 and L_2. The optimal values of the control parameters for the different control architectures are obtained by imposing constraints on the corresponding control parameters in the optimization problem. For example, to optimize for an architecture where the fuel-rate (m_{out}) does not depend on the fuel combustor temperature (Tf), we impose the constraint $0 \leq G_1 \leq 0$. Or to optimize for an architecture where the heat-sink efficiency f does not depend on the fuel-tank temperature (T), we impose the constraint $0 \leq L_1 \leq 0$.

In the rest of this chapter, the different possible control architectures are represented by a numeric code. For example, the numeric code 0100 represents the architecture with the following constraints imposed on the control parameters:

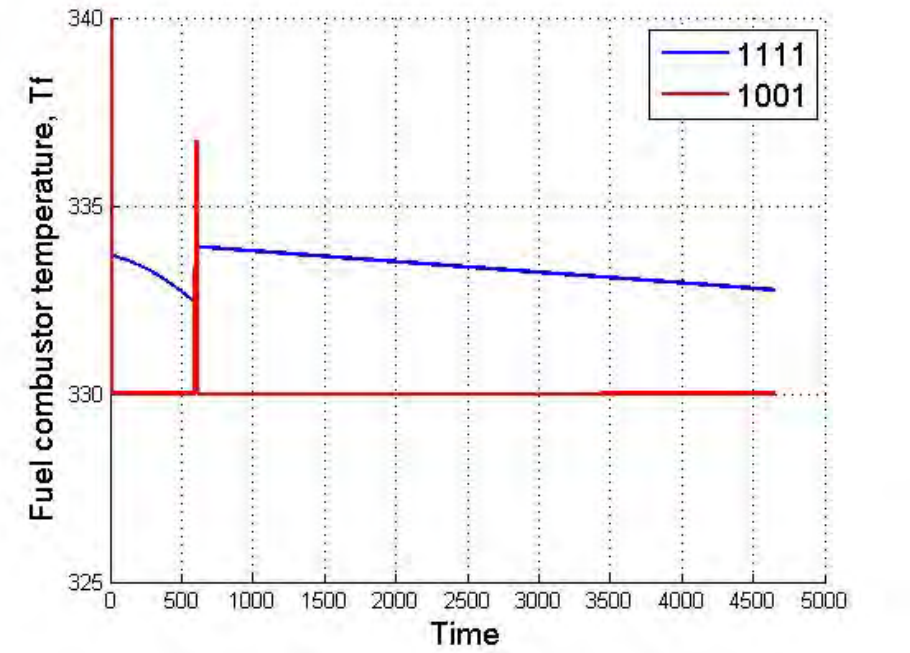
$$\begin{aligned}
0 \leq G_1 \leq 0 \\
0 \leq G_2 \leq \infty \\
0 \leq L_1 \leq 0 \\
0 \leq L_2 \leq 0
\end{aligned} \tag{9.12}$$

In other words, the architecture with numeric code 0100 is such that the fuel-flow rate has feedback dependency on the heat-sink efficiency (f), but all other feedback dependencies are absent. We optimize the control parameters for nominal values of the flying time and taxing time ($\Delta_{taxi} = 600, \Delta_{fly} = 4050$) and for the same system parameters shown in Table 7.1, but with $T_{set} = 330$ and $\bar{T} = 280$. The optimal values of the control parameters for the different control architectures are shown in Tables 9.1 and 9.2. Figure 9.5 shows the trajectories for the states of the TMS (T and Tf) corresponding to the optimal values of the control parameters obtained for architectures 1111 and 1001. Figure 9.6 shows the corresponding trajectories for the controlled states of the TMS (m_{out} and f).

As discussed in Chapter 9, we use stochastic automata to capture the failure modes of the controllers. The stochastic automate to capture the failure modes of the controllers are composed with

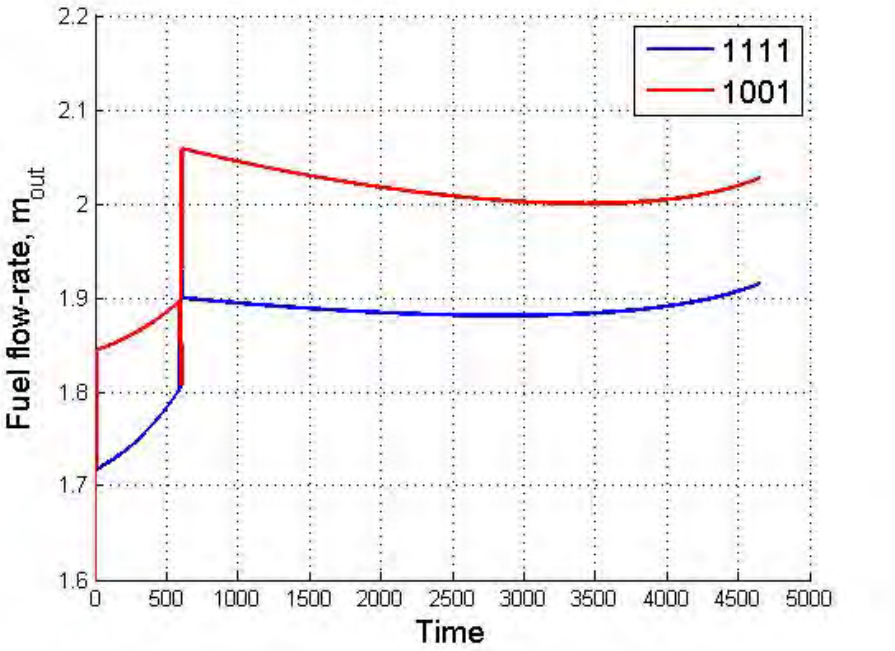


(a) Fuel-tank temperature, T

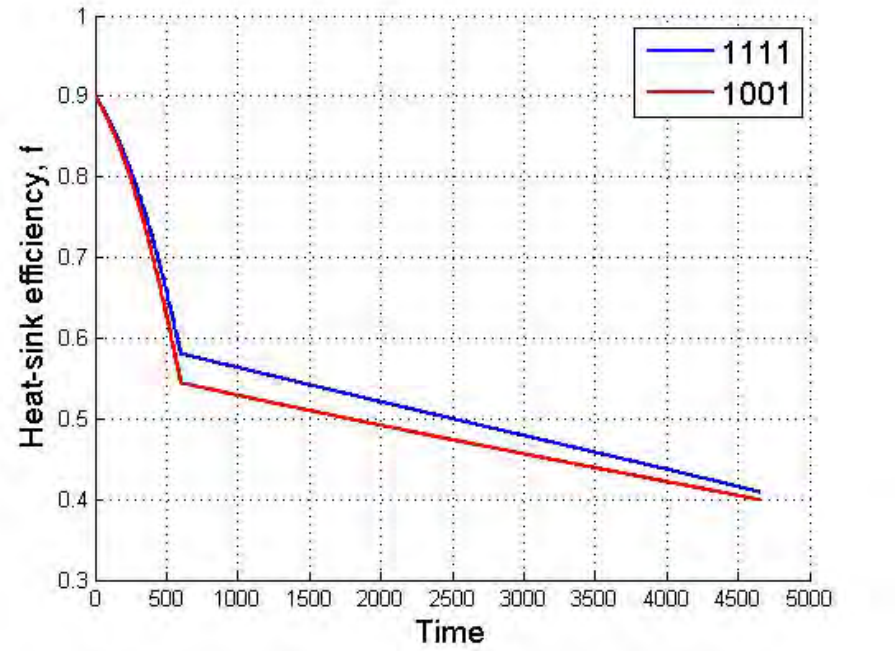


(b) Fuel combustor temperature, Tf

Figure 9.5: Plots of the fuel-tank temperature (T) and fuel combustor temperatures (Tf) for the optimal values of the control parameters obtained for architectures 1111 and 1001.



(a) Fuel flow-rate, m_{out}



(b) Heat sink efficiency, f

Figure 9.6: Plots of the fuel flow-rates (m_{out}) and the heat sink efficiency (f) for the optimal values of the control parameters obtained for architectures 1111 and 1001.

Architecture	G_1	G_2	L_1	L_2
1000	0.0130	0.0	0.0	0.0
1001	0.0134	0.0	0.0	0.0016
1010	0.0130	0.0	0.0	0.0
1011	0.0134	0.0	0.0	0.0016
1100	0.0159	0.0473	0.0	0.0
1101	0.0160	0.0658	0.0	0.0017
1110	0.0159	0.0488	0.0	0.0
1111	0.0160	0.0658	0.0	0.0017

Table 9.1: Optimal control parameters in the taxi mode for different control architectures.

Architecture	G_1	G_2	L_1	L_2
1000	0.0	0.0	0.0	0.0
1001	0.016047	0.0	0.0	0.000157
1010	0.0	0.0	0.0	0.0
1011	0.016047	0.0	0.0	0.000157
1100	0.042945	0.669275	0.0	0.0
1101	0.032050	0.216040	0.0	0.000257
1110	0.038578	0.560197	0.0005	0.0
1111	0.032050	0.216040	0.0	0.000257

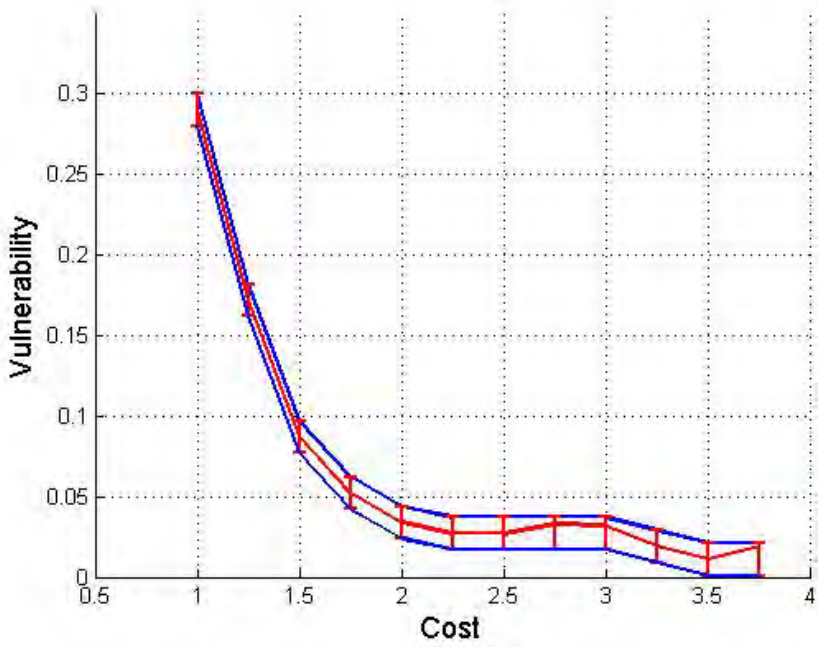
Table 9.2: Optimal control parameters in the flying mode for different control architectures.

the stochastic hybrid system described in Equation 9.4. We then analyze the composed stochastic hybrid model to study how the vulnerability of the system changes with respect to parameters in the failure models. In this study, we consider only permanent faults as described in Chapter 9. In the OP mode, the controller is operating normally as intended by design. In the FAIL mode, the controller is faulty and its outputs are set to a nominal value. The controller switches from the OP mode to the FAIL mode with a probability λ. For implementing a processing element that fails with a certain probability λ, there is a certain cost associated with it. As a representative cost, we use the relation

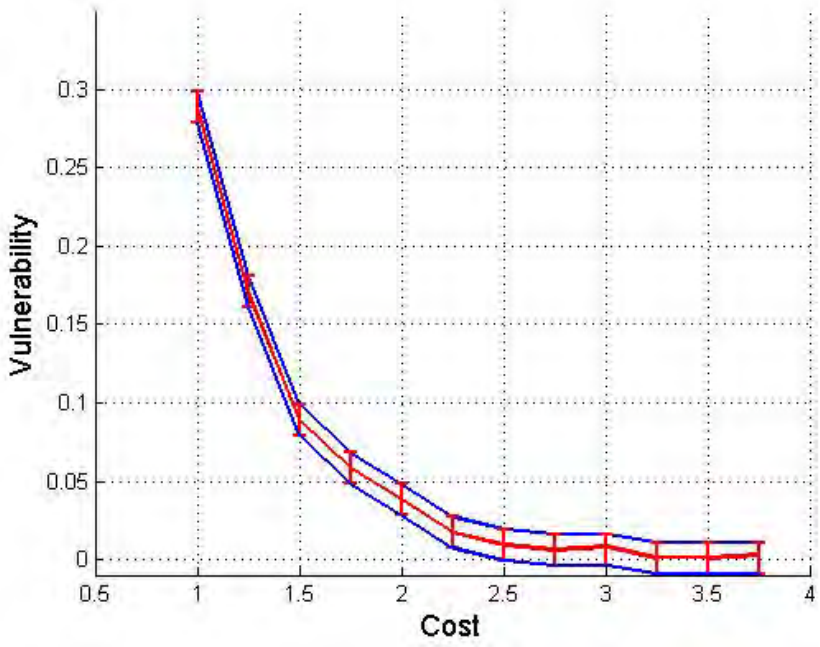
$$H(\lambda) = \frac{\log(\lambda)}{\log(\lambda_0)} \tag{9.13}$$

In other words, the probability of the processing element to fail decreases in an exponential fashion as its cost increases. We are interested in studying how the vulnerability of the system changes as the cost for the implementation of a platform is increased (or reliability is increased).

For this particular case study, the condition for checking vulnerability is as follows. We compute the fraction of time spent by the trajectory (T^k, Tf^k) outside the set $(250, 310) \times (300, 360)$. If this fraction of time is greater than 0.01, we declare the system to be vulnerable. We generate many realizations for the trajectories of the composed stochastic hybrid system by Monte-Carlo simulations. We estimate the vulnerability as the fraction of the number of realizations for which the system was vulnerable. Figure 9.7 shows how the vulnerability of the system changes with respect to the cost of the processing element. From the plots in 9.7, one can figure out the cost of the processing element required to guarantee that the vulnerability will be below a prescribed threshold. These plots are for architectures 1001 and 1010. These architectures perform as well as the architecture 1111 in terms of performance optimality and vulnerability. Since these architectures have fewer links than the architecture 1111, they are definitely more desirable.



(a) Architecture 1001



(b) Architecture 1010

Figure 9.7: Vulnerability vs. cost plots for architectures 1001 and 1010.

9.5 Summary and future steps

In this chapter, we have described a two-step procedure to explore the effects of the control architecture and reliability of platform processing elements on the vulnerability of the thermal management system. The control architecture exploration involves the enumeration of all possible architectures and optimizing for the control algorithm that can be implemented on each control architecture. In the future, it would be desirable to replace this enumeration step by more efficient approaches where the critical control links are discovered automatically. Simultaneous optimization of the control architecture and the platform architecture is challenging in general. As of now, it is possible to design the platform features (like the processing elements) only once the control architecture has been fixed. It would be desirable to develop approaches where the platform features can be jointly optimized for with the control architecture.

Bibliography

[1] "http://www.mrmc-tool.org."

[2] "U.s. standard atmosphere," U.S. Government Printing Office, Tech. Rep., 1976. [Online]. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19770009539_1977009539.pdf

[3] *Specification and refinement of probabilistic processes*, Jul. 1991. [Online]. Available: http://dx.doi.org/10.1109/LICS.1991.151651

[4] *Joint 13th IFIP WG 6.1 International Conference on Formal Methods for Open Object-based Distributed Systems and 31th IFIP WG 6.1 International Conference on FORmal TEchniques for Networked and Distributed Systems (FMOODS/FORTE), 6-9 June 2011, Reykjavik, Iceland*, ser. Lecture Notes in Computer Science, vol. 6722. Springer, 2011.

[5] A. Abate, A. D'Innocenzo, and M. D. Benedetto, "Approximate abstractions of stochastic hybrid systems," *IEEE Transactions on Automatic Control*, 2009, provisionally Accepted.

[6] A. Abate, J. Katoen, J. Lygeros, and M. Prandini, "Approximate model checking of stochastic hybrid systems," *European Journal of Control*, 2010, provisionally Accepted.

[7] A. Abate, "Probabilistic reachability for stochastic hybrid systems: Theory, computations, and applications," Ph.D. dissertation, University of California, Berkeley, November 2007. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/440.html

[8] R. Alur, T. A. Henzinger, and P. H. Ho, "Autoatic symbolic verification of embedded systems," *IEEE Transactions on Software Engineering*, vol. 22, pp. 181–201, 1996.

[9] C. Baier, B. Engelen, and M. Majster-Cederbaum, "Deciding bisimilarity and similarity for probabilistic processes," *Journal of Computer and System Sciences*, vol. 60, pp. 187–231, 2000.

[10] C. Baier, H. Hermanns, and J.-P. Katoen, "Probabilistic weak simulation is decidable in polynomial time," *Inf. Process. Lett.*, vol. 89, pp. 123–130, February 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=975948.975952

[11] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximate symbolic model checking of continuous-time markov chains," in *Proceedings of the 10th International Conference on Concurrency Theory*, ser. CONCUR '99. London, UK: Springer-Verlag, 1999, pp. 146–161. [Online]. Available: http://portal.acm.org/citation.cfm?id=646734.701464

[12] ——, "Approximate symbolic model checking of continuous-time markov chains (extended abstract)," 1999, p. 781. [Online]. Available: http://www.springerlink.com/content/t22lk0et8c9r40c4

[13] A. Benveniste, B. Caillaud, and R. Passerone, "Multi-viewpoint state machines for rich component models," in *Model-Based Design of Heterogeneous Systems*.

[14] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, "Multiple viewpoint contract-based specification and design," in *Proceedings of the Software Technology Concertation on Formal Methods for Components and Objects (FMCO'07)*, ser. Revised Lectures, Lecture Notes in Computer Science, vol. 5382. Amsterdam, The Netherlands: Springer, October 2008.

[15] L. Benvenuti, A. Ferrari, L. Mangeruca, E. Mazzi, R. Passerone, and C. Sofronis, "A contract-based formalism for the specification of heterogeneous systems," in *Proc. Forum Specification, Verification and Design Languages FDL 2008*, 2008, pp. 142–147.

[16] H. A. P. Blom, G. J. Bakker, and J. Krystul, "Probabilistic reachability analysis for large scale stochastic hybrid systems," in *46th IEEE Conference on Decision and Control, New Orleans, USA*. IEEE, December 2007, pp. 3182–3189.

[17] J. Bogdoll, L. M. F. Fioriti, A. Hartmanns, and H. Hermanns, "Partial order methods for statistical model checking and simulation," in *FMOODS/FORTE*, ser. Lecture Notes in Computer Science, vol. 6722. Springer, June 2011, pp. 59–74.

[18] F. Borrelli, *Constrained Optimal Control of Linear and Hybrid Systems*, ser. Lecture Notes in Control and Information Sciences. Springer, 2003, vol. 290.

[19] M. Bujorianu and J. Lygeros, "Toward a general theory of stochastic hybrid systems," *Lecture Notes in Control and Information Sciences (LNCIS)*, vol. 337, pp. 3–30, 2006. [Online]. Available: http://control.ee.ethz.ch/index.cgi?page=publications;action=details;id=2596

[20] L. P. Carloni, R. Passerone, A. Pinto, and A. L. Angiovanni-Vincentelli, "Languages and tools for hybrid systems design," *Foundation and Trends in Electronic Design Automation*, vol. 1, no. 1/2, pp. 1–193, 2006.

[21] C. G. Cassandras, D. L. Pepyne, and Y. Wardi, "Optimal control of a class of hybrid systems," *IEEE Transactions on Automatic Control*, vol. 46, no. 3, pp. 398–415, 2001.

[22] B. Delahaye, B. Caillaud, and A. Legay, "Probabilistic contracts: A compositional reasoning methodology for the design of stochastic systems," in *Application of Concurrency to System Design (ACSD), 2010 10th International Conference on*, june 2010, pp. 223 –232.

[23] B. Delahaye and B. Caillaud, "A Model for Probabilistic Reasoning on Assume/Guarantee Contracts," INRIA, Research Report RR-6719, 2008. [Online]. Available: http://hal.inria.fr/inria-00337538/en/

[24] B. Delahaye, B. Caillaud, and A. Legay, "Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or non-deterministic aspects," *Form. Methods Syst. Des.*, vol. 38, pp. 1–32, February 2011. [Online]. Available: http://dx.doi.org/10.1007/s10703-010-0107-8

[25] M. Dellnitz and O. Junge, *Set Oriented Numerical Methods for Dynamical Systems*. World Scientific, 2002, pp. 221–264.

[26] A. D??Innocenzo, A. Abate, M. D. D. Benedetto, and S. S. Sastry, "Approximate abstractions of discrete-time controlled stochastic hybrid systems," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, December 2008, pp. 221–226. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/443.html

[27] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis, "Multi-objective model checking of Markov decision processes," in *Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, ser. LNCS, O. Grumberg and M. Huth, Eds., vol. 4424. Springer, 2007, pp. 50–65.

[28] G. Frehse, "Phaver: algorithmic verification of hybrid systems past hytech," *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 3, pp. 263–279, 2008.

[29] G. Froyland, "Extracting dynamical behaviour via markov models," in *Nonlinear Dynamics and Statistics*, A. Mees, Ed., Newon Institute, Cambridge. Birkhauser, 1998, pp. 283–324.

[30] R. German, D. Logothetis, and K. S. Trivedi, "Transient analysis of markov regenerative stochastic petri nets: A comparison of approaches," in *In 6-th International Conference on Petri Nets and Performance Models - PNPM95*. IEEE Computer Society, 1995, pp. 103–112.

[31] H. Giese, "Contract-based component system design," in *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8*, ser. HICSS '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 8051–. [Online]. Available: http://portal.acm.org/citation.cfm?id=820264.820479

[32] H. Gimbert, "Pure stationary optimal strategies in markov decision processes," in *Proceedings of the 24th annual conference on Theoretical aspects of computer science*, ser. STACS'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 200–211. [Online]. Available: http://portal.acm.org/citation.cfm?id=1763424.1763449

[33] D. Hartfiel, *Markov set-chains*, ser. Lecture notes in mathematics. Springer, 1998. [Online]. Available: http://books.google.com/books?id=79wZAQAAIAAJ

[34] S. Hedlund and A. Rantzer, "Optimal control of hybrid systems," in *IN PROCEEDINGS OF THE 38TH IEEE CONFERENCE ON DECISION AND CONTROL*, 1999, pp. 3972–3977.

[35] T. A. Henzinger, P. H. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," *Software Tools for Technology Transfer*, vol. 1, pp. 110–122, 1997.

[36] J. Hu, J. Lygeros, and S. Sastry, "Towards a theory of stochastic hybrid systems," in *In Third International Workshop on Hybrid Systems: Computation and Control*. Springer, 2000, pp. 160–173.

[37] J. P. Katoen, D. Klink, M. Leucker, and V. Wolf, "Three-valued abstraction for continuous-time markov chains," in *CAV'07: Proceedings of the 19th international conference on Computer aided verification*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 311–324.

[38] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen, "The Ins and Outs of The Probabilistic Model Checker MRMC," in *Quantitative Evaluation of Systems (QEST)*. IEEE Computer Society, 2009, pp. 167–176, www.mrmc-tool.org.

[39] A. J. Kleywegt, A. Shapiro, and T. Homem-De-Mello, "The sample average approximation method for stochastic discrete optimization," *SIAM J. Optim.*, vol. 12, no. 2, pp. 479–502, 2001.

[40] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Assume-guarantee verification for probabilistic systems," in *Proc. 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, ser. LNCS, J. Esparza and R. Majumdar, Eds., vol. 6105. Springer, 2010, pp. 23–37.

[41] A. Lasota and M. C. Mackey, *Chaos, Fractals and Noise*, ser. Applied Mathematical Sciences. Springer, 1994.

[42] J. Mattingly, W. Heiser, and D. Pratt, *Aircraft engine design*, ser. AIAA education series. American Institute of Aeronautics and Astronautics, 2002, no. v. 1. [Online]. Available: http://books.google.com/books?id=2Wy5rpdm3DMC

[43] M. F. Neuts, *Matrix-geometric solutions in stochastic models : an algorithmic approach / Marcel F. Neuts*. Johns Hopkins University Press, Baltimore :, 1981.

[44] D. A. Parker, "Implementation of symbolic model checking for probabilistic systems," Tech. Rep., 2002.

[45] A. Pinto and S. Krishnamurthy, "Developing design tools for uncertain systems in an industrial setting," in *Allerton*, September 2010.

[46] B. Plateau and K. Atif, "Stochastic Automata Network of Modeling Parallel Systems," *IEEE Transactions on Software Engineering*, vol. 17, no. 10, pp. 1093–1108, 1991.

[47] S. Prajna, A. Jadbabaie, and G. Pappas, "Stochastic safety verification using barrier certificates," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 1, dec. 2004, pp. 929 –934 Vol.1.

[48] M. Prandini and J. Hu, "Application of reachability analysis for stochastic hybrid systems to aircraft conflict prediction," *Automatic Control, IEEE Transactions on*, vol. 54, no. 4, pp. 913 –917, april 2009.

[49] D. Riley, X. Koutsoukos, and K. Riley, "Reachability analysis for stochastic hybrid systems using multilevel splitting," in *Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 460–464. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00602-9_35

[50] T. Schouwenaars, B. Mettler, E. Feron, and J. P. How, "Robust motion planning using a maneuver automaton with built-in uncertainties," in *In Proc. 2003 American Control Conference*, 2003, pp. 2211–2216.

[51] A. Wald, "Sequential Tests of Statistical Hypotheses," *The Annals of Mathematical Statistics*, vol. 16, no. 2, pp. 117–186, 1945.

[52] A. Wchter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006.

[53] D. N. Xu, G. Gössler, and A. Girault, "Probabilistic contracts for component-based design," in *8th Int. Symp. on Automated Technology for Verification and Analysis (ATVA 2010)*, ser. Lecture Notes in Computer Science. Springer, 2010, to appear. [Online]. Available: http://gallium.inria.fr/~naxu/research/atva10.pdf

[54] H. L. S. Younes and R. G. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," *CAV*, vol. 2404, pp. 223–235, 2002.

[55] ——, "Statistical probabilistic model checking with a focus on time-bounded properties," *Inf. Comput.*, vol. 204, no. 9, pp. 1368–1409, 2006.

[56] P. Zuliani, A. Platzer, and E. M. Clarke, "Bayesian statistical model checking with application to stateflow/simulink verification," in *Proceedings of 13th International Conference on Hybrid Systems: Computation and Control*, 2010.

[57] ——, "Bayesian statistical model checking with application to simulink/stateflow verification," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, ser. HSCC '10. New York, NY, USA: ACM, 2010, pp. 243–252. [Online]. Available: http://doi.acm.org/10.1145/1755952.1755987